OPTIMIZATION FOR STOCHASTIC, PARTIALLY OBSERVED SYSTEMS USING A SAMPLING-BASED APPROACH TO LEARN SWITCHED POLICIES

BY

SALVATORE JOSEPH CANDIDO

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Doctoral Committee:

Professor Seth A. Hutchinson, Chair
Professor Sean P. Meyn
Assistant Professor Todd P. Coleman
Assistant Professor Enlu Zhou

ABSTRACT

We propose a new method for learning policies for large, partially observable Markov decision processes (POMDPs) that require long time horizons for planning. Computing optimal policies for POMDPs is an intractable problem and, in practice, dimensionality renders exact solutions essentially unreachable for even small real-world systems of interest. For this reason, we restrict the policies we learn to the class of switched belief-feedback policies in a manner that allows us to introduce domain expert knowledge into the planning process. This approach has worked well for the systems on which we have tested our approach, and we conjecture that it will be useful for many real-world systems of interest.

Our approach is based on a method like value iteration to learn a switching law. Because the POMDP problem is intractable, we use a Monte Carlo approximation to evaluate system behavior and optimize a switching law based on sampling. We explicitly analyze the sensitivity of expected cost (performance) with respect to perturbations introduced by our approximations, and use that analysis to avoid approximation errors that are potentially disastrous when using the computed policy. We demonstrate results on discrete POMDP problems from the literature and a resource allocation problem modeled after a team of robots attempting to extinguish a forest fire.

We then utilize our approach to build two algorithms that solve the minimum uncertainty robot navigation problem. We demonstrate that our approach can improve on techniques in the literature in terms of solution quality by demonstrating our results in simulation. Our second approach utilizes information-theoretic heuristics to drive the sampling-based learning procedure. We conjecture that this is a useful direction towards an efficient, general stochastic motion planning algorithm.

ACKNOWLEDGMENTS

I thank those at the University of Illinois with whom I worked outside of the research domain, but who influenced not only my thought process and technical skills but also my views of what an engineer is and does. Although there are too many people to name, in particular I would like to acknowledge Tamer Basar, P.R. Kumar, Mark Spong, Dan Roth, David Forsyth, Andreas Cangellaris, and Dan Block.

I would be remiss not to acknowledge my colleagues - in particular, James Davidson, Sourabh Bhattacharya, Kenton McHenry, Akash Kushal, Han-Ul Yoon, Bobby Gregg, Steven Kloder, Rafael Murrieta-Cid, Tim Bretl, Devin Bonnie, Yong-Tae Kim, and Adam Tilton. I am lucky to have worked with all of them, and luckier to consider many of them friends to this day.

My committee members, Seth Hutchinson, Sean Meyn, Todd Coleman, and Enlu Zhou, were invaluable resources throughout the later stages of my Ph.D. program. I thank them for both their patience and willingness to help me improve my work.

I acknowledge and thank my adviser, Seth Hutchinson, for his guidance and sage counsel through every step of my graduate career. In both technical and professional matters, his advice has always been a beacon guiding me towards this goal and beyond.

I thank my friends for keeping me connected and grounded during my graduate work, and my family for keeping me focused and purposeful. Without the consistent encouragement and support of Sal, Marilyn, Joseph, Michael, and David Candido, this dissertation would have not been possible.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

We present a method for learning feedback policies for processes belonging to the class of partially observable Markov decision processes (POMDPs) that have large state, observation, and control spaces. Learning the optimal feedback policy for a POMDP is known to be hard [1]. Excluding a few cases where the models are of a special form, e.g., linear quadratic Gaussian systems, in practice an approximation to this policy is required. Our approach is based on explicitly restricting the policies considered by the algorithm to the set of switched policies. Thus, the policy learning procedure entails optimizing a switching law and not choosing individual controls. This immediately results in not only a reduction of the search space, but also a problem decomposition that allows for sensible utilization of principled, standard approximation techniques. Specifically, approximations can be utilized that preserve the "structure" of the problem, which is key in preventing catastrophic approximation errors that can lead to learning a policy that performs poorly. This problem decomposition also allows domain-specific, expert knowledge to be seamlessly inserted into the learning procedure. This serves to counteract the possibly damaging effects of an ad hoc reduction of the search space.

Before describing this work in technical detail in later chapters, this chapter addresses the general motivation for working on this problem and the specific motivation of the approach chosen. We will discuss the advantages of choosing the POMDP framework as a system model, the optimization challenges the model creates, and our general approach to overcoming these challenges. The chapter closes with a summary of the contributions contained in this dissertation and an outline of the remainder of the contents.

1.1 Why Model Systems with POMDPs?

A large volume of research in a number of communities has been focused on computing optimum policies for POMDPs. Nevertheless, current approaches are still unsatisfactory for use on many problems in practice. Many real-world systems modeled as POMDPs remain well beyond the reach of even supercomputers using standard methods, and yet developing techniques to solve these problems is necessary for deployment of "intelligent" or "smart" systems. Thus, this area or research is not only interesting, but also widely applicable and of immediate interest to a range of disciplines. Improving the scalability of optimization techniques is one of the most important problems confronting us, because efficiency is becoming increasingly important in the face of rising demand for natural resources, power, labor, and computation. Much theory exists, but computational and mathematical hurdles have prevented the transfer of classical optimization techniques to large stochastic problems. The work in this dissertation is a step towards circumventing these hurdles to develop a general, principled approach that can be applied to very large systems.

A POMDP is an excellent tool for engineering models in a variety of areas. It is primarily composed of a state process that is governed by stochastic difference (or differential) equations. But, unlike a Markov decision process (MDP), the system state is not assumed to be perfectly and immediately measurable by the controller or any other external process. This is very often an appropriate model for many real-world phenomena, as we are often able to model systems very succinctly if many parameters are known, but typically few of those parameters will actually be perfectly measurable.

While the POMDP framework has many uses, this research is focused on algorithms and techniques that enable autonomy. Classically, autonomy is equated with robots, but the field is also essential for systems that are sufficiently large or fast to require integration of intelligent behavior. These types of systems are typically characterized by an enormous number of degrees of freedom or significant uncertainty.

Although the results of this dissertation for specific engineering problems are focused on a few key applications, POMDPs can be used to model systems ranging from a single autonomous robot to large-scale population dynamics. The optimization procedure we present has the potential to be used

to improve the autonomy of cyber-physical systems used in real-world applications, e.g., unmanned aerial vehicles, search and rescue robots, self-driving cars, and humanoid-like robots. Aside from these standard robotics applications, it could also be used as a starting point for optimization of transient[1] behaviors of many large systems, e.g., the power grid, transportation systems, air traffic control, the environment, health care, and epidemiological models.

1.2 Computational Limitations Confine POMDP Optimization

While POMDPs are excellent for modeling, they have historically been of limited use in optimization for large systems. Learning the optimum feedback policy for a POMDP with respect to an objective function, is known to be hard both in terms of complexity class and "in practice" computational cost. This is due to two fundamental aspects inherent in the POMDP optimization problem. The first is dimensionality of the *belief*, the characterization of what is known about the state process. The second is the exponential growth of the number of beliefs that must be examined to decide the optimum controls for the feedback policy.

Since we cannot observe the state directly and the occurrences of events are governed by probability functions, the condition of the system is characterized by a probability distribution over the possible values of the current state. A transformation, described by Bayesian filtering equations, can be used convert the system description to a new set of differential or difference equations. A posterior probability distribution over the state space conditioned on the information vector is used as the state variable of the transformed system. This probability distribution and de facto state variable is referred to as a belief. This characterization essentially lifts[2] the problem description into a higher-dimensional space where the evolution of the system has less uncertainty. There is a significant representational cost inherent

[1] Typically for these very large systems a steady state behavior is optimized and measures are taken to ensure that this operating point is met. With appropriate tools, greater efficiencies could be attained by changing the operating point and optimizing transient dynamics based on external factors.

[2] The lifting process is described in detail in Chapter 2.

in this transformation, as the number of possible states governs the number of degrees of freedom of the probability function over state. Thus, the dimensionality of the lifted system will be equal to the size of the state space for the original state process. This implies that the dimensionality of the "system" we must control, i.e., the probability function, will rapidly become extremely large or infinite. Furthermore, uncertainty often exacerbates an already present dimensionality problem caused by a large number of degrees of freedom present in many systems of interest.

The second aspect comes from the observation model, the component of a POMDP that accompanies the state process. The evolution of the belief process, i.e., the sequence of probability functions that represent the likelihoods of the state process as it evolves, is affected by measurements correlated with the true system state. Only one measurement per time step is recorded and influences the POMDP's belief process. However, at optimization time (in advance of actual use of the policy) the sequence of measurements the belief process will receive and use is a random object. Suppose, for example,[3] there are n discrete observations that the belief process may receive; then at stage t there will be n^t possible sequences of observations. This implies the number of beliefs the optimization procedure must examine is $O(n^t)$.

For large models with complicated dynamics, often the dimension of the belief will be high (because the state space will be large) and the optimization horizon must be long to achieve quality planning results. In order to make progress on these models, we meet both of the primary complexity hurdles of POMDP optimization head-on. This reveals an uncomfortable fact: although the POMDP is an excellent engineering model for many systems of practical interest, we typically cannot hope to compute optimum policies for the systems we model. It is the complexity challenges, and not the theoretical difficulty of the POMDP problem, that constitute the primary obstacle to more widespread use of the POMDP framework.

In fact, techniques for computing optimum policies have been known for a significant length of time. Applying the aforementioned lifting transformation, a POMDP can be represented as an MDP and techniques from control theory and optimization theory can be applied. This was explicitly demonstrated in [2] for finite state, finite-horizon problems and similar results have

[3]This argument will be clear in more generality after the discussion of *hyperbelief* in Chapter 2.

been demonstrated for other classes of systems, e.g., continuous state spaces are discussed in [3]. In the case of [2], the optimum policy can be computed by using a dynamic programming approach whose termination condition is the terminal cost of the system. However, to guarantee an optimum solution using this algorithm (and others like it) requires exponential complexity with respect to optimization horizon.

Many research communities have brought their own flavor to building approximation approaches to the POMDP optimization problem. Specific methods will be discussed in Section 3.4, but we limit discussion here to only classes of approaches. *Finite horizon approaches* seek to limit the growth in the number of beliefs that must be considered by optimizing for only a limited number of stages into the future. Unfortunately, the quality of the policy can be degraded by having an insufficient forward-looking view of how the system will evolve. Intuitively, this can be illustrated by the "ultimate" finite horizon approach: the greedy policy. Furthermore, if the policy is to be generated in advance of use, it may be infeasible to compute the policy at every belief, and determining the specific subset of beliefs where the policy must be computed can be expensive. Thus, it makes sense for the policy to be computed on the fly as the process moves. Significant progress has been made in augmenting this type of approach using a *Monte Carlo point-based approach*. Essentially, the main principle is the same, but the sample paths[4] used to compute the best control and length of horizon are chosen using Monte Carlo sampling. While these methods can significantly boost performance by allocating computation commensurately with the likelihood of events that may occur, they can still suffer from nearsightedness. The driving principle of both finite horizon and point-based methods is that the near future is significantly more important than what will happen later, and that later can be accounted for in due course.

The opposite side of the spectrum is a *roadmap-style approach*. The main goal of a roadmap approach is to consider the coarse evolution of the system by leaving the local aspects of control, i.e., specific controls at specific stages, to a local planner. As it becomes more and more clear that the short-sighted methods will not scale to many systems of interest, roadmap-style approaches have recently been gaining popularity in the literature (and this dissertation

[4]A sample path is the sequence of beliefs the process will encounter and is discussed further in Chapter 2.2.3.

describes one such method). Of course, it should not be claimed that this approach is strictly better. Because of the primary benefit of a roadmap-style approach, i.e., coarse evolution of the system allowing a long horizon view of the system, an important step can be missed locally or the planner may never have the resolution necessary to optimize certain systems. Keeping the optics metaphor, roadmap-style approaches can suffer from farsightedness, which can also lead to policies that are disastrously poor.

1.3 Specific Challenges of Optimizing Complicated POMDPs

As discussed in the previous section, there are trade-offs and drawbacks to every approximation one might attempt. When asking what approximations and approaches are most useful as we continue to address increasingly larger and more complex systems, it is important to recognize that these questions are typically best answered with respect to the problem at hand. We are interested in system models with interesting dynamics, non-trivial planning solutions, many degrees of freedom, high levels of uncertainty, and non-trivial control and observation spaces.

- A system with **interesting dynamics** refers to a system with a process model that may be nonlinear or include discontinuities. The system itself may be complicated, or perhaps the system is simple but the process model is complicated due to the presence of constraints, e.g., obstacles in a robot navigation problem. These properties imply that system trajectories are not trivial to analyze and may require simulation to compute.

- Systems with **non-trivial planning solutions** are systems where a short planning horizon will not suffice to achieve the system objective, or where it is not easy to show that a short-sighted strategy is, in fact, optimal. This is typically a consequence of a nontrivial objective in conjunction with interesting dynamics. This combination makes it difficult to write an objective function whose gradient guides the system to the goal everywhere. Lack of such an objective function typically implies we will require a long planning horizon.

- When considering systems with **many degrees of freedom** and **high levels of uncertainty**, we are assured of a problem with large dimensionality. Engineering models for large, complicated systems will typically have a large state space. Of course, the number of degrees of freedom of the system model will be degrees of freedom of the POMDP optimization problem. Additionally, the amount of uncertainty affects the complexity of the probability functions necessary to predict the possible condition of the system, which increases the dimensionality of the problem. Consequently, the fact that there will be large dimensionality implies that simulation of the system will be expensive.

- Generally speaking, systems with **non-trivial control and observation spaces** entail control and observation spaces that are either continuous or discrete with high cardinality. The number of controls and observations dictate the branching factor of the optimization search tree, which is rooted at the initial belief and has directed edges emanating from each node that correspond to control-observation pairs. A t-step walk through the tree is equivalent to a t-stage possible trajectory of the POMDP. The size of the control space dictates the branching factor of the optimization procedure (the number of possible choices one can make) and multiplying by the size of the observation space dictates the complexity of what must be optimized (the number of places where one may have to make a choice). Thus, for non-trivial control and observation spaces, choosing the optimum control for even a short horizon can be computationally difficult.

To illustrate this branching factor difficulty, consider an example of a team of 10 robots moving on a two-dimensional grid; a problem with small size in a robotics context. If each robot can move to any of its eight neighboring cells, the set of controls has 2^{30} possibilities to evaluate. If the cost function is nontrivial and we have no additional way to prune this set, a brute-force approach will require checking 2^{30} controls to compute a (very shortsighted) 1-step optimal policy. In fact, if we sample this set uniformly without replacement we will need 2^{28} samples in order to achieve only a 0.25 probability of finding the optimal one-step control. Thus, challenges presented by a non-trivial control space dictate that it can be fruitful to choose controls, in the short term, based on a local heuristic.

The other multiplier in the optimization search tree branching factor is the number of elements in the observation space. Unlike the control, where the controller can explicitly ignore some possibilities, explicitly ignoring certain observations will not make them go away. However, even for a fixed policy (removing the branching based on controls) there will be an exponential growth in the number of leaf nodes of the optimization search tree as the depth increases. We will not be able to computationally handle this growth. Challenges presented by a nontrivial observation space demand that we focus computational efforts on higher likelihood sequences of observations. This can be done, for example, using a Monte Carlo approach to exploring the optimization search tree.

Since we must typically simulate POMDP evolution due to interesting system dynamics, and this simulation is expensive due to large problem dimensionality, it is prudent to re-use POMDP simulation as much as possible, and when possible, trade computer memory for computer processing. Furthermore, if one can modify simulations to be applicable for a range of parameters it will significantly reduce the necessary computation. This also implies that it may take a very long time to reach the optimum solution, so we must develop approaches that build partial solutions on the way to converging at the optimum solution. Since the dimension of the belief will be high, we may need to approximate the belief in order to maintain a tractable representation of the POMDP state.

Finally, the branching factor of the optimization search tree in combination with the requirement of a long planning horizon implies we will not be able to optimize one stage at a time. Specifically, the class of systems described here dictates a roadmap-type of approach and, for many systems of interest, the expected cost of use can be greatly reduced by optimizing policies at a coarse level.

The nature of the approach seems clear from the argument above, but as previously mentioned, roadmap methods can suffer from farsightedness and this type of approach should not be applied in an ad hoc manner. Specifically, the sequences of controls over which the policy is optimized should have a meaningful correspondence with characteristics of the system, and the approach should not amount to simply optimizing for a coarse-grained or (excessively) model reduced version of the original model. The next section addresses how these particular problems can, in some cases, be avoided.

1.4 A Structurally Motivated Approach to the POMDP Optimization Problem

Although the problem requirements dictate we use a roadmap-style approach to optimization, pragmatically we cannot apply canonical roadmap-style algorithms without expecting approximation error and mistakes in the optimization process. We can circumvent these problems, at least partially, by using an approach motivated by considering the problem structure of a POMDP. The key insight is that, for many engineering models, small perturbations to trajectories in the majority of the trajectory space produce small perturbations to cost. This smoothness principle is, of course, not universally true, and there will be regions of the space where cost changes rapidly, and surfaces where cost changes discontinuously. When lifting these differential models into transition laws for the belief and hyperbelief space (via Bayesian filtering), the smooth and discontinuous portions of the model are simply lifted into a higher dimensional space, and the structure of the original model is preserved. Thus, it is essential to choose approximations that carefully preserve, and in some cases exploit, this problem structure.

The structural properties that implicitly exist in most engineering models justify use of techniques such as sampling-based planning and Monte Carlo approximation of system behavior; sensitivity analysis can be used to verify the validity of the approximations. We have exploited the structure present in the equations that lift the stochastic model into deterministically evolving probability spaces to develop a scalable, anytime approach that allows expert knowledge to be integrated into the planning process. We have created several algorithms that have been successful, and in some cases successful beyond the state of the art approaches in the literature. This approach has worked well for several systems, and we conjecture that it will be useful for many more real-world systems of interest.

Our POMDP optimization approach performs a coarse optimization of the policy and incrementally refines the policy using an anytime algorithm,[5] but maintains a fine-grained analysis of effect of the policy on the system. The approach is based on the application of domain, i.e., problem-specific, knowledge. In practice, this problem-specific knowledge is often available

[5] An anytime algorithm is an algorithm that can, at any point of its execution, return a partial answer whose quality depends on the amount of computation.

in the form of locally optimal trajectories or other quality policies that are effective over a few time steps or in certain situations. The main idea of our approach is to synthesize a set of these policies into a composite policy that is effective in all situations the system may encounter. This is a type of hierarchical approach to policy generation.

Of course, while a reduction of the search space will be effective for reducing the time until solution, there is no guarantee that the quality of the solution will not also be reduced. Reduction of solution quality in the general case is hard to quantify. While this is an important area of future investigation, we do not address it in this dissertation. Instead, we justify the utility of the method by demonstrating excellent results on a number of POMDPs in different problem domains. In the worst case, the results presented in this dissertation are commensurate with the best reported results in the literature. Beyond that, we have demonstrated results on very large (relative to other results presented in the literature) discrete systems, and demonstrated that our method is useful for stochastic motion planning.

The primary advantage of this method is that, because these local policies guide the system evolution over multiple stages, we can evaluate the effects of a particular policy choice many stages into the future. This is essential for finding effective policies for many systems. To be clear, we gain this advantage by restricting our policy space; i.e., we explicitly disregard trajectories not generated by sequences of local policies. However, if the local policies are of sufficient quality we demonstrate that we can achieve excellent results with a vastly reduced amount of computation. Based on experimental evidence, injecting a small amount of domain knowledge appears to be a prudent addition to the planning procedure.

Our method accelerates the planning process by performing a value backup over multiple stages at every iteration, not just evaluating a single control used for a single time step. This allows a more depth-first exploration of the policy search tree, which typically leads to discovery of quality policies much more quickly than does a breadth first, expansive search. However, the mechanics of this procedure require us to overcome a number of difficulties. Because we are evaluating the POMDP's behavior many stages in the future, we characterize system trajectories in the *hyperbelief space*, the space of probability functions over the belief space, and develop approximation techniques to cope with the exponential explosion of support points in

that space. Furthermore, in order to gain a tangible, practical advantage from this approach, we restrict our method to techniques that allow us to achieve *temporal abstraction*. This means that once we simulate the effect of a local policy, we must parameterize the simulation for re-use in optimization at later iterations and we must be able to re-use the parameters in $O(1)$ time with respect to the number of stages covered by the initial simulation.

Our approach builds a policy on the belief space, not the hyperbelief space. However, we use analysis of the POMDP's evolution in hyperbelief space extensively during the planning procedure. Since we explicitly plan in the belief space, there are no difficulties in translating our analysis of the POMDP's evolution into a belief-feedback policy. We construct an approximation of the value function at sampled points in the belief space. Then, we use a policy iteration-like approach to gradually improve the policy, which allows us to formulate our approach as an anytime algorithm. This is desirable, as our method is primarily targeted at systems where simulation and planning may take significant time, and the policy may need to be used before planning can be fully completed. We use a sampling-based approach where sampling occurs in the space defined by the product of the belief space and the set of local policies. Thus, our method samples where we will expend computational effort to attempt to improve the policy, not points in the belief space for the planner to attempt to reach. This is essential, because in general a POMDP is not controllable in either the belief space or the hyperbelief space, so attempting to reach arbitrary points is typically futile.

Our method builds a directed graph in the belief space that represents the hierarchical evolution of the POMDP; i.e., vertices correspond to beliefs and groups of edges correspond to the behavior of the POMDP under a local policy. By gradually expanding this graph and applying policy improvement, the method is able to incrementally build a policy of demonstrable quality. Although this is conceptually similar to policy iteration, it is not just the canonical algorithm on an atypical graph. Because local policies are executed over a varying number of stages (one local policy may take significantly different numbers of stages based on the observations it receives), we cannot simply abstract them as macro-actions for a smaller POMDP. Our method looks multiple stages into the future, but after initial analysis, avoids re-computing the effects of local policies stage-wise every time we optimize. This becomes particularly challenging when combined with spatial

abstraction, i.e., re-using previous system simulation for new points in the belief space, based on a perturbation (sensitivity) analysis of the POMDP's behavior.

As mentioned above, our approach allows for the inclusion of domain-specific, specialist knowledge. Specifically, the modes of the switched policy can be chosen to perform tasks in a way that are known to be useful components of optimal trajectories. This method is a trade-off between the computational gains of excessive model reduction[6] and optimization of a feedback policy one control at a time. The search space reduction, structure-preserving approximations, and inclusion of domain knowledge, in concert, contribute to a method that is both scalable and effective.

We have demonstrated the results of this method on discrete POMDP "benchmark" problems, and a very large discrete robot coordination problem. We use this method to demonstrate a minimum uncertainty navigation approach for general robot systems.

1.5 Summary of Contributions

1. We present a method with roadmap-style approach to learning policies for very large, complicated systems modeled with POMDPs. Our proposed method is capable of integrating domain-specific knowledge into the planning process.

2. We present, for discrete POMDP systems, an explicit analysis of problem structure that justifies the approach.

3. We demonstrate that our method works in practice on both discrete benchmark problem and continuous robot navigation problems, as well as on a discrete POMDP significantly larger than others previously reported in the literature.

4. We apply our method to solve the minimum uncertainty navigation problem for robot systems.

[6] Of course, model reduction should always be applied where it is useful and the approximation is appropriate.

1.6 Dissertation Outline

In Chapter 2, we provide a general overview of the POMDP model. We begin with a discussion of the model and its evolution in the state, belief, and hyperbelief spaces. We then discuss Monte Carlo approximation of the system evolution

In Chapter 3, we discuss learning optimal policies for POMDPs. This chapter begins by mathematically defining the POMDP optimization problem and the necessary accompanying concepts in Section 3.1. In Section 3.2, we discuss dynamic programming and the value function, which is a ubiquitous tool in optimization theory. This discussion is primarily a prerequisite to discussing the main classes of approach to the POMDP optimization problem in Section 3.3. This section concludes with a short discussion of the intractability of these methods for large systems. Thus, in Section 3.4 we discuss some specific optimization methods broken into a rough taxonomy of approaches.

Chapter 4 contains the core discussion of our approach. In this chapter, we begin by formally describing the class of switched policies in Section 4.1. We then discuss the evolution of a POMDP under a local policy, which we term an expansion and explicitly define in Section 4.3. This computation is useful because it characterizes the evolution of the system from a point in the belief space until the next switching time. Unfortunately, the expansion operator is computationally expensive, so we discuss approximation in Section 4.3.2. Once these tools are in place, we use them to construct an algorithm to learn a switched policy in Section 4.4. Because this algorithm is computationally impractical, we conclude by proposing a sampling-based approximation algorithm in Section 4.4.2, which is one of our main results. We conclude with a discussion of our algorithm in Section 4.5.

We then present results of this method on discrete state, observation, and control POMDP problems in Chapter 5. These results to demonstrate the viability and scalability of our method. We begin in Section 5.1 by discussing our approach as applied to several POMDP benchmark problems, which is done to demonstrate that our results are commensurate to other POMDP solvers on established problems. We then present results on a more interesting system, a resource allocation problem modeled after a team of robots attempting to extinguish a forest fire, in Section 5.2.

Chapter 6 presents analysis that explains why our method works in practice. We compute sensitivity functions for the expansion operator that is defined in Chapter 4 for discrete state systems. We postulate that a similar analysis is possible for continuous state POMDPs. We briefly discuss the benefits and drawbacks of including this sensitivity analysis into POMDP optimization methods.

In Chapter 7 we apply our POMDP optimization technique to the minimum uncertainty robot navigation application. We briefly survey robot navigation in Section 7.1.1 and discuss why minimum uncertainty robot navigation is hard. We present specific stochastic robot navigation techniques in Section 7.1.2. With the requisite background, we then discuss our general approach to attack the problem in Section 7.2. We then present our two specific methods to solve this problem. The first, discussed in Section 7.3, is based on using a roadmap heuristic and is mainly presented as a comparison to other techniques in the literature. The second, presented in Section 7.4, incorporates information theory heuristics. We believe this second method is an important step in building a general purpose minimum uncertainty navigation planner for general robot systems.

Finally, in Chapter 8 we provide a concluding discussion of the work outlined in this dissertation and suggest some interesting new directions for future research.

CHAPTER 2

POMDP SYSTEM DYNAMICS

In this chapter, we describe and discuss the mathematical details of a generic system described by a POMDP. A system model is specified in terms of conditional probabilities governing the state and observation processes. However, this description also induces processes that evolve in two levels of posterior conditional probability functions. One can step down each of these levels by specifying a particular value of a random object. This alludes to the reason that we must understand all three characterizations of a POMDP. Specifically, as the system itself evolves it has access to the current state. However, as an external observer (like a controller) tries to estimate that state, it does not have access to the state, so it *filters* the conditional posterior probability function over state, which is referred to as a belief. When learning a policy (before the process occurs), there is no access to the sequence of observations. Thus, the POMDP is best described in terms of the conditional posterior probability function over belief.

The goal of this chapter is to describe all three levels of POMDP system dynamics. We begin with a brief review of the POMDP model, primarily to establish notation. We start by discussing a POMDP description characterized by state transition probabilities and observation probabilities, and then lifting the POMDP description to transition equations in the belief space. When learning a policy, we must consider uncertainty in future observations. Thus, we lift the POMDP description into a higher dimensional functional space that considers all observations with nonzero probability, the hyperbelief space. We then present a brief section to solidify the links between these three POMDP descriptions. Finally, we discuss approximations to these descriptions.

2.1 POMDP Model

A POMDP is comprised of a state space $\mathcal{X}$, control space $\mathcal{U}$, set of observations $\mathcal{Y}$, and the conditional probability functions that correspond to the evolution of the state and observation processes, i.e., $\mathrm{f}_{\boldsymbol{x}_{t+1}|\boldsymbol{x}_t,u_t}(x_{t+1} \mid x_t, u_t)$ and $\mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{x}_t,u_{t-1}}(y_t \mid x_t, u_{t-1})$, where $u_t, u_{t-1} \in \mathcal{U}$, $\boldsymbol{x}_{t+1}, \boldsymbol{x}_t \in \mathcal{X}$, $\boldsymbol{y}_t \in \mathcal{Y}$. In this dissertation, we consider POMDPs governed by difference and not differential equations, and the subscripted t indicates the stage number or time step of the process. The notation above indicates that the state transition and observation likelihoods are described by probability density functions (pdf's). Of course, the state space, or observation space, could be discrete, continuous, or mixed[1] and thus pdf's or probability mass functions (pmf's) should be used as appropriate. The probabilistic state transition and observation laws imply that, even if the current state x_t is known a priori, the state at the next stage $\boldsymbol{x}_{t+1}$ and the observation $\boldsymbol{y}_t$ will be a random objects.[2]

Although it is not strictly required to specify a POMDP, in many engineering applications the state transition probabilities arise from a process model $f : \mathcal{X} \times \mathcal{U} \times \mathcal{N} \to \mathcal{X}$ and the observation probabilities arise from an observation[3] model $h : \mathcal{X} \times \mathcal{U} \times \mathcal{M} \to \mathcal{Y}$. The sets $\mathcal{N}$ and $\mathcal{M}$ are sets of possible random values injected into the process and observation models. Depending on the model, these sets may again be composed of a discrete set of elements or a continuum of values. The process model is

$$\boldsymbol{x}_{t+1} = f(x_t, u_t, \boldsymbol{n}_t)$$

where $n_t \in \mathcal{N}$. The mapping f is deterministic, but $\boldsymbol{n}_t$ is a random variable drawn from a stationary probability distribution $\mathrm{f}_{\boldsymbol{n}}$ defined over $\mathcal{N}$. Similarly, we have an observation model

$$\boldsymbol{y}_t = h(x_t, u_{t-1}, \boldsymbol{m}_t)$$

where $\boldsymbol{m}_t \in \mathcal{M}$ and is drawn from the stationary distribution $\mathrm{f}_{\boldsymbol{m}}$. The combi-

[1] Typically in the literature, a space is referred to as mixed if the parameters describing a point in the space have both discrete and continuous elements.

[2] A random object $\boldsymbol{r}$ is denoted with a bold font, and an instantiation or value of this random object will be written without the bold font, e.g., r.

[3] This is often equivalent to a *sensor model*, and thus used interchangeably, in the robotics literature.

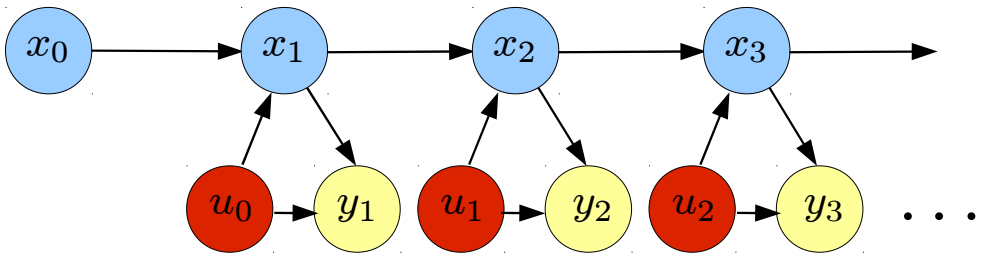


Figure 2.1: POMDP Conditional Dependences

nation of the deterministic function f and probability distribution $\mathrm{f}_{\boldsymbol{n}}$ induces the state transition probabilities, and the same goes for the observation probabilities with respect to h and $\mathrm{f}_{\boldsymbol{m}}$. We refer to a model given in terms of a set of state transition and observation probabilities as being specified *explicitly*. Alternatively, we refer to a model given in terms of process and observation models (often plus constraints) as being specified *implicitly*.

An implicit specification, if possible, is often advantageous over an explicit one. Firstly, the number of parameters necessary to specify the model will often be fewer than in an explicit representation. For example, consider a queue with random arrivals where $x_t, u_t, n_t \in \mathbb{Z}^+$ and n_t is generated by a Poisson process. This can be modeled by the process

$$\boldsymbol{x}_{t+1} = \max(x_t - u_t + \boldsymbol{n}_t, 0).$$

Representing this process requires only a specification of this function and storage of the rate parameter. Any transition probability can then be computed on the fly. In contrast, we will need to store $2(|\mathcal{X}| - 1) \times |\mathcal{U}|$ numbers to explicitly represent this system, which will be particularly difficult if the queue length is unbounded. As in the above example, the implicit representation can easily be used to generate an explicit representation, given sufficient computation and storage. However, moving in the other direction is a fundamental problem in learning theory and, for many classes of f, is a hard problem. Another reason the implicit specification is desirable is because the equations representing the process analyzed, and useful structure of the system, e.g., regions of continuous variation and discontinuities, can often be extracted.

Consider Figure 2.1, the conditional dependence chart of a POMDP. The

evolution of a POMDP can be thought of as a three-step process for each stage. First, a control u_t is applied to the system and the state transitions from x_t to x_{t+1} with probability $\mathrm{f}_{\boldsymbol{x}_{t+1}|\boldsymbol{x}_t,u_t}(x_{t+1} \mid x_t, u_t)$. Second, an observation y_{t+1} is received with probability $\mathrm{f}_{\boldsymbol{y}_{t+1}|\boldsymbol{x}_{t+1},u_t}(y_{t+1} \mid x_{t+1}, u_t)$. This observation does not affect the trajectory of the state process in any way, but can be used by an external process to infer information about the state process. Finally, based on a causal control policy, u_{t+1} is chosen and these steps are repeated for stage $t+2$.

Since external systems, such as a controller, cannot directly distinguish the current state of the system, we can characterize the condition of a POMDP based on the set of all information that is known. This information is composed of the initial state estimate, and the sets of observations received and controls sent to the system. Let $y_{1:t} = \{y_1, y_2, \cdots, y_t\}$ and $u_{0:t} = \{u_0, u_1, \cdots, u_t\}$, i.e., the sets of observations and controls up to stage t. Then, the *information state* I_t at stage t is the set of directly known information from the initial stage through stage t, i.e.,

$$I_t = \{\mathrm{f}_{\boldsymbol{x}_0}, y_{1:t}, u_{0:t-1}\}$$

where $\mathrm{f}_{\boldsymbol{x}_0}$ is the probability function describing the likelihoods of the initial state. The *information space* $\mathcal{I}_t$ is the set of all possible information states at stage t.

The information state is typically not a convenient representation of the condition of a POMDP, as its size grows with every subsequent stage. However, the posterior probability function over the state space conditioned on the information vector can store an equivalent characterization of the current condition of a POMDP. This probability function is typically a more useful representation, and in the next section we discuss how it can replace the information vector.

2.2 POMDP Evolution in the Belief Space

The conditional posterior probability function on the state space is commonly referred to as a *belief* (and is also occasionally called the *hyper-state*). We

will denote the belief[4] at stage t as b_t, i.e.,

$$b_t(x) := \mathrm{f}_{\boldsymbol{x}_t|\boldsymbol{I}_t}(x|I_t)$$

The space of all possible probability distributions on the state space is referred to as the *belief space*, denoted $\mathcal{P}_b$. The belief was shown in [4] to be a sufficient statistic of the information state.[5] Specifically, this means that the information vector provides no information about the state that the belief does not also provide, i.e.,

$$\mathrm{f}_{\boldsymbol{x}_t|\boldsymbol{b}_t}(x|b_t) = \mathrm{f}_{\boldsymbol{x}_t|\boldsymbol{b}_t,\boldsymbol{I}_t}(x|b_t, I_t)$$

for all x. This is intuitively obvious from the fact that the belief is, itself, conditioned on the information vector, so additional conditioning on the information vector is redundant. This also implies that the belief can be used as a more compact representation of the condition of a POMDP, and we develop equations in the next section that specify how the belief evolves with respect to the system model, controls, and observations.

In some cases, the belief is finitely parameterizable. For example, in the case of conjugate priors the belief at stage $t+1$ will belong to the same family of distributions as the belief at stage t. An example of this is a linear Gaussian system, where the belief will always be a Gaussian distribution (if the process starts with Gaussian initial belief) and the belief can always be parameterized with a constant size mean vector and covariance matrix. For POMDPs with a discrete, finite state space, a belief can be represented conveniently as a column vector of dimension $|\mathcal{X}|$, and the belief space is a $|\mathcal{X}|$-1 dimensional simplex (taking into account the constraint that every belief vector must sum to one).

2.2.1 Belief Propagation using Bayesian Filtering

Bayesian filtering is the Bayesian approach to maintaining a probability distribution of an unknown random object as information describing the random object, such as stochastic measurements, is incrementally arriving. It is

[4] In practice, we often cannot represent the exact pdf and instead use an approximation. We will use $\hat{b}_t$ to refer to an approximation of the exact belief b_t.

[5] A more expository discussion can be found in [5].

sometimes referred to, more descriptively, as recursive Bayesian estimation. The Bayesian filtering equation for belief propagation is well known, e.g., [6], so in this section we simply present a decomposition that highlights the correspondence between particular portions of the equation and elements of a POMDP. However, for readers unfamiliar with Bayesian filtering, a complete derivation is provided is Appendix A.

For the purpose of this section, we will presume the observation at the current stage is given. Thus, the discussion of this section is most applicable to the case where the filtering equations are being used to estimate the state of a system as it evolves. In Section 2.2.3, we discuss the belief process beyond the assumption that it is simply an estimation tool.

The *belief process operator* T_u updates the belief according to the behavior associated with applying the process model, given a particular $u \in \mathcal{U}$. If b_t is the current belief state, the quantity $b_{t+1|t}$ is the Bayesian prediction, i.e.,

$$b_{t+1|t} = T_{u_t} b_t$$

The operator can be defined point-wise

$$b_{t+1|t}(x_{t+1}) = \int_{x_t \in \mathcal{X}} \mathrm{f}_{\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, u_t}\left(x_{t+1} \mid x_t, u_t\right) b_t(x_t) dx_t \tag{2.1}$$

Note that we refer to $b_{t+1|t}$ as a belief for convenience, but this is in fact an abuse of notation. Although $b_{t+1|t}$ can be parameterized identically to a belief, it is actually the pdf of $\boldsymbol{x}_{t+1}$ conditioned on I_t and u_t, not I_{t+1}.

The *belief observation operator* $O_{y,u}$, for $y \in \mathcal{Y}$, serves to update the belief according to the most recent observation received. The operator updates the predicted belief, e.g.,

$$b_{t+1} = O_{y_{t+1}, u_t} b_{t+1|t}$$

and the point-wise state likelihood under this operator is

$$\frac{1}{\eta_{t+1}} b_{t+1}(x) = \mathrm{f}_{\boldsymbol{y}_{t+1}|\boldsymbol{x}_{t+1}, u_t}\left(y_{t+1} \mid x_{t+1}, u_t\right) b_{t+1|t}(x) \tag{2.2}$$

The scalar η_{t+1} is a normalizing constant; i.e., it ensures the integral of (2.2) over $\mathcal{X}$ is equal to one and the belief remains a valid pdf. The composite operation over all x is $O_{y,u}$ and, in concert with normalization, is referred to as Bayesian update.

Thus, the *belief transition operator* for the POMDP is

$$b_{t+1} = \frac{O_{y_{t+1},u_t} T_{u_t}}{\int_{x \in \mathcal{X}} O_{y_{t+1},u_t} T_{u_t} b_t(x) dx} b_t = \eta_{t+1}\ O_{y_{t+1},u_t} T_u b_t = \phi_{y_{t+1},u_t} b_t \qquad (2.3)$$

This development has assumed a continuous state space. If the state is discrete or mixed, the equations are similar but summation replaces integration. Note that the normalizing constant is equal to the inverse of the conditional likelihood of observation y_{t+1}.

2.2.2 Transition Operators for Finite, Discrete POMDPs

For a POMDP with a discrete and finite state space, i.e., $\mathcal{X} = \{x^1, x^2, \cdots, x^{|\mathcal{X}|}\}$, the belief and transition operators have a special structure. Specifically, the transition operators will be $|\mathcal{X}| \times |\mathcal{X}|$ matrices and the belief will be a length $|\mathcal{X}|$ (column[6]) vector. Since the belief is a pmf over a state space with $|\mathcal{X}|$ elements, the belief will be a vector with every component being a real number in $[0, 1]$ and all components summing to one. Thus, the belief space is the $(|\mathcal{X}| - 1)$-dimensional simplex.

The belief process and observation operators are $|\mathcal{X}| \times |\mathcal{X}|$ matrices with special structure. The belief process operator is represented by a set of left stochastic matrices where entries correspond to probabilities of transitioning between specific states. The matrix corresponding to control u will be T_u and can be generated using

$$[T_u]_{ij} = \mathrm{p}_{\boldsymbol{x}_{t+1}|\boldsymbol{x}_t,u_t}\left(x_{t+1} = x^i \mid x_t = x^j, u_t = u\right)$$

where $[A]_{ij}$ indicates the i, j entry of the matrix A. Thus, entry i, j of T_u is the probability of transitioning from the j^{th} state to the i^{th} state, and left-multiplying b_t by T_u corresponds to Bayesian prediction. The belief observation operator is a set of diagonal matrices, where the matrix corresponding to observing y after performing control u is denoted $O_{y,u}$ and

$$[O_{y,u}]_{ii} = \mathrm{p}_{\boldsymbol{y}_t|\boldsymbol{x}_t,u_t}\left(y_t = y \mid x_t = x^i, u_t = u\right)$$

[6]We choose the representation of the belief to be a column vector to be consistent with control systems literature.

Analogously, left-multiplication of $O_{y,u}$ corresponds to Bayesian update.

If we define the vector $\mathbf{1}$ to be a vector of length $|\mathcal{X}|$ where every entry is one, then the probability of the observation y at stage t under the distribution b_t is $\mathbf{1}'O_{y,u}T_u b_t$. Thus, the belief transition can be computed with a matrix multiplication

$$b_{t+1} = \frac{O_{y,u}T_u}{\mathbf{1}'O_{y,u}T_u b_t} b_t = \phi_{y,u} b_t \tag{2.4}$$

The notation v' refers to the transpose of the vector v.

2.2.3 Belief Process

It is important to note that the evolution of a POMDP (for a fixed sequence of controls and observations) in the belief space is completely specified by the belief transition function. Although there is uncertainty in the state of the POMDP (the belief is a probability function over the state space), the belief transition functions are completely deterministic given a sequence of controls and observations. However, predicting the evolution of the process in the future can only be done probabilistically as the sequence of observations will not be known a priori. Thus, since we have the observation as a random object, as a consequence of (2.3) the next stage belief will also be a random object, i.e.,

$$\boldsymbol{b}_{t+1} = \phi_{\boldsymbol{y}_{t+1},u_t} b_t$$

Particular sequences of observations and controls define a *sample path* for the belief process through the belief space. Along specific sample path, i.e., for a known sequence of information states, the belief evolves deterministically according to the belief transition operator. But the sample path trajectory is unknown a priori because the content of the information vector will only be known probabilistically for future stages.

Consider Figure 2.2, which shows the conditional dependence relationships for the belief process. A box is placed around the state process to indicate that it cannot be directly observed by an external process, e.g., the filtering equations for the belief process. If the state process is ignored, the conditional dependence chart indicates that the belief process is an MDP with belief acting as the "state" of the process, ϕ acting as a process model, and the observation acting as the perturbation (source of uncertainty).

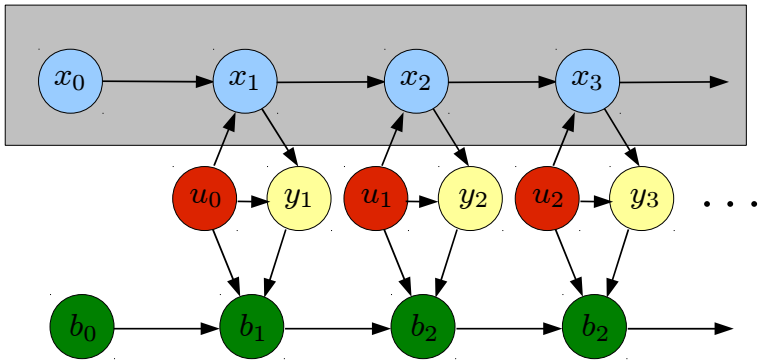


Figure 2.2: Belief Process Conditional Dependences

This connection is fundamental and motivates characterizing the possible beliefs a POMDP may reach probabilistically, much as we characterize the possible states in which the process may reside with a belief. We discuss this in the next section.

2.3 Hyperbelief Evolution of a POMDP

When described in terms of a system's state space, the evolution of the state of a POMDP is governed by a set of transition probabilities that describe the effect of the process model. If, instead, the system is described in terms of the belief space, the evolution of the system can be modeled as a MDP using (2.4) as the process model. This corresponds to lifting the system description from a lower dimensional state space to a higher dimensional belief space. Most POMDP optimization algorithms operate, and approximate the system, at the level of the belief space. For a sample path, the POMDP's evolution can be completely captured by a single trajectory through the belief space. When the system is in operation it follows a sample path, but the particular sample path will only be known a posteriori, since the next observation is a random object, with only a prior distribution on its value.

If we want to analyze the complete behavior of a POMDP in the future, e.g., to evaluate the possible effect of a control policy, we must consider all sample paths that have nonzero probability. The lifting process can be continued an additional step by lifting the belief space to the higher dimensional space of probability functions that can be defined on the belief space. We

refer to this lifted space as the *hyperbelief space*. In the hyperbelief space, the system evolves deterministically, thus eliminating (in some sense) the explicit consideration of uncertainty during the planning process, at the expense of a much higher representational cost.

Our approach utilizes analysis in the hyperbelief space. Our motivation for this explicit characterization is three-fold. First, to evaluate the behavior of the system many stages into the future, it is useful to characterize the transition function in the probability space over $\mathcal{P}_b$. Second, by explicitly characterizing this transition function we can approximate it using standard approximation methods, e.g., using a Monte Carlo sampling approach. Finally, we can use this formalism to evaluate cost metrics beyond the standard expected cost formulation.

Before defining a hyperbelief, we must briefly discuss the concept of a policy. A policy π is a specification, given some information, of what control should be used next. We will introduce policies in detail in Chapter 3 (and significant discussion subsequently throughout the remainder of this dissertation), but for the purpose of this section it will suffice to consider a policy to be a mapping from belief to control.

A *hyperbelief* β is a functional defined over $\mathcal{P}_b$. It satisfies the properties $\beta : \mathcal{P}_b \to \mathbb{R}^+$ and $\int_{\mathcal{P}_b} \beta(b_t) db_t = 1$. The hyperbelief at stage t is the conditional probability density of b_t based on an initial hyperbelief and policy, i.e.,

$$\beta_t(b) = \mathrm{f}_{\boldsymbol{b}_t}\left(b \mid b_0, \pi\right)$$

and we denote the *hyperbelief space* $\mathcal{P}_\beta$ as the set of all admissible hyperbeliefs. For notational convenience, we suppress the explicit mention of the starting location and policy on which the probability function is conditioned, as they are typically clear from context. If the observation space is countable, then the hyperbelief will be a weighted sum of Dirac delta functions centered on the set of beliefs that can be reached at stage t. This implies that, for this case, the hyperbelief will be a pmf but the domain of the pmf is nontrivial to characterize. For simplicity, we will always denote the hyperbelief as a pdf, but it is important to recognize the hyperbelief will have a particular structure for many classes of POMDP systems.

The same recursive Bayesian estimation principles govern the evolution of the hyperbelief, and we can use them to derive the *hyperbelief transition*

function. The hyperbelief transition function $\Phi_\pi : \mathcal{P}_\beta \to \mathcal{P}_\beta$, under π, governs the evolution of the POMDP in the hyperbelief space, i.e., $\beta_{t+1} = \Phi_\pi \beta_t$. The hyperbelief transition function is defined point-wise by

$$(\Phi_\pi \beta_t)(b_{t+1}) = \int_{\mathcal{P}_b} \int_{\mathcal{Y}} \delta\left(b_{t+1} - \phi_{y_{t+1},\pi(b_t)} b_t\right) \mathrm{f}_{\boldsymbol{y}_{t+1}|\boldsymbol{b}_t,\pi}\left(y_{t+1}|b_t,\pi\right) \beta(b_t)\, dy_{t+1}\, db_t \tag{2.5}$$

where δ is the Dirac delta function and the functional notation $(\Phi_\pi \beta)(b)$ indicates the pdf value assigned to the belief b by the hyperbelief $\Phi_\pi \beta$. Equation (2.5) is derived in Appendix B.

Notice that because we marginalize over all observations, the POMDP evolution in the hyperbelief space is deterministic. We have thus characterized a POMDP by a set of nonlinear, deterministic equations evolving in an infinite-dimensional space. These equations have a particular structure which is induced by the structure of the process and observation models of the state process combined with the canonical Bayesian filtering equations.

While useful analytically, operating in the hyperbelief space produces challenges analogous to those when operating in the belief space; e.g., discrete observation systems exhibit exponential growth in the number of reachable beliefs as the number of stages elapsed increases. Representing and computing the exact hyperbelief evolution of a POMDP is, in many cases, an impractical proposition. At best, i.e., $\mathcal{Y}$ discrete and finite, computing β_t suffers from the same intractability problem as computing the reachable belief space. At worst, i.e., $\mathcal{Y}$ continuous, we are faced with a continuous function over $\mathcal{P}_b$ whose representation requires an increasing number of parameters as the process evolves. We address representation of the hyperbelief for simulation purposes in the next section.

2.4 Approximating POMDP Evolution

A common tool for sequential estimation in cases where the estimate function is complicated is the *particle filter* [7], which is a sequential Monte Carlo method. A particle filter represents the posterior probability of a latent variable by a set of samples. Each sample consists of a value the latent variable may have and a weight that corresponds to the (approximate) likelihood of that value. We utilize particle filtering throughout this dissertation, both to

Input: $\hat{b}_t := \{(x_t^{(i)}, w_t^{(i)})\}_{i=1}^N$, y_{t+1}, u_t
Output: $\hat{b}_{t+1} := \{(x_{t+1}^{(i)}, w_{t+1}^{(i)})\}_{i=1}^N$

$\hat{b}_{t+1|t} \leftarrow \emptyset$
foreach $(x_t^{(i)}, w_t^{(i)}) \in \hat{b}_t$ **do**
  $x_{t+1|t}^{(i)} \leftarrow f(x_t^{(i)}, u_t, n_t)$ where n_t is sampled from $\mathrm{f}_{\boldsymbol{n}}$
  $w_{t+1|t}^{(i)} \leftarrow \mathrm{f}_{\boldsymbol{y}_{t+1}|\boldsymbol{x}_{t+1},u_t}\left(y_{t+1}|x_{t+1|t}^{(i)}, u_t\right) \mathrm{f}_{\boldsymbol{x}_{t+1}|\boldsymbol{x}_t,u_t}\left(x_{t+1|t}^{(i)} \mid x_t^{(i)}, u_t\right)$
  $\hat{b}_{t+1|t} \leftarrow \hat{b}_{t+1|t} \cup \{(x_{t+1|t}^{(i)}, w_{t+1|t}^{(i)})\}$
end
$\hat{b}_{t+1} = resample(\hat{b}_{t+1|t})$

Algorithm 2.1: Particle Filter

approximate hyperbelief evolution and, in some cases, to approximate the belief of systems when it is expensive to perfectly represent it. While there is a great wealth of algorithms and analysis for particle filtering methods, the discussion in this section is simply a remedial review of the canonical particle filtering algorithm and a functional description of utilizing it to approximate the hyperbelief, which is non-standard in the literature.

Essentially, a particle filter estimate is a pdf of the form

$$\hat{b}_t = \sum_{i=1}^{N} w_t^{(i)} \delta\left(x - x_t^{(i)}\right) \tag{2.6}$$

where the samples are chosen to represent a subset of the support of the original pdf and

$$\sum_{i=1}^{N} w_t^{(i)} = 1$$

The remainder of the support is discarded, and the estimate is propagated forward. With a sufficient number of samples, a particle filter can estimate arbitrary distributions and, under certain conditions, will approach the true Bayesian estimate (pdf) as the number of samples goes to infinity. In practice, we must be careful to provide a sufficient number of particles or the algorithm will suffer from sample impoverishment and the estimate will fail to be consistent.

Because the intuition is much simpler and the notation represents the

actual implementation of the estimate, we colloquially write

$$\hat{b}_t = \left\{(x_t^{(1)}, w_t^{(1)}), \cdots, (x_t^{(N)}, w_t^{(N)})\right\}$$

as an equivalent expression to (2.6). The interpretation is that the particle filter is composed of a set of particles, i.e., tuples of state and importance factor.

Algorithm 2.1 implements a basic particle filtering method. We begin with a set of particles $(x_t^{(i)}, w_t^{(i)})$ representing the estimate at stage t in conjunction with control applied to and (known) observation received from the system. The sample states in the particles are "pushed forward" one stage by sampling the evolution of the system from the state specified by the initial particle. The likelihood of these new samples is then evaluated in light of the observation, and new particles are constructed with weights indicating the relative likelihood of each sample with respect to the likelihoods of the samples in all other particles. A survival of the fittest resampling (with replacement) step is then used to allocate resources to the more important particles. This roughly corresponds to approximating the probability distribution over the state with more precision at points of greater likelihood.

Although typically applied to estimate a belief (i.e., choosing the state to be the latent variable), we can also apply this technique to estimate a hyperbelief using the belief process as the latent variable. The use of particle filtering in this way is called *hyper-particle filtering* and was proposed in [8]. The nomenclature *hyper-particle* refers to the fact that each particle is a weight and belief, not a weight and state. As an aside, this discussion highlights one of the key advantages of moving to a hyperbelief-based representation of the condition of a POMDP. While the inherent complexity remains, this representation allows us to apply principled, standard approximation techniques to estimate POMDP evolution.

To curb the exponential growth of support points in $\mathcal{P}_\beta$, we represent β_t with an approximated $\hat{\beta}_t$ with a finite number of weighted support points. The approximated hyperbelief is then

$$\hat{\beta}_t = \left\{(b_t^{(1)}, w_t^{(1)}), \cdots, (b_t^{(N_\mathrm{b})}, w_t^{(N_\mathrm{b})})\right\} \tag{2.7}$$

Note that $\hat{\beta}_t \in \mathcal{P}_\beta$, but is not the exact conditional probability distribution

Input: $\hat{\beta}_t := \{(b_t^{(i)}, w_t^{(i)})\}_{i=1}^N$, π
Output: $\hat{\beta}_{t+1} := \{(b_{t+1}^{(i)}, w_{t+1}^{(i)})\}_{i=1}^N$

$\hat{\beta}_{t+1} \leftarrow \emptyset$
foreach $(b_t^{(i)}, w_t^{(i)}) \in \hat{\beta}_t$ **do**
  $b_{t+1}^{(i)} \leftarrow \phi_{y_{t+1},u_t} b_t^{(i)}$ with y_{t+1} is sampled from $\mathrm{f}_{\boldsymbol{y}_{t+1}|\boldsymbol{b}_t,u_t}\left(\cdot|b_t^{(i)}, \pi(b_t^{(i)})\right)$
  $w_{t+1|t}^{(i)} \leftarrow \mathrm{f}_{\boldsymbol{y}_{t+1}|\boldsymbol{b}_t,u_t}\left(y_{t+1}|b_t^{(i)}, \pi(b_t^{(i)})\right)$
  $\hat{\beta}_{t+1} \leftarrow \hat{\beta}_{t+1} \cup \{(b_{t+1|t}^{(i)}, w_{t+1|t}^{(i)})\}$
end
$\hat{\beta}_{t+1} = resample(\hat{\beta}_{t+1})$

Algorithm 2.2: Hyper-particle Filtering

of b_t. We use Algorithm 2.2 to propagate the belief through the POMDP process model. This is similar to the typical particle filter application, but note there are a few significant differences. Firstly, instead of specifying a specific control we must specific a policy that prescribes the appropriate control for each belief. Also, the process model, in standard particle filtering, is replaced with the belief transition function in hyper-particle filtering. Since the belief process with unknown observations is an MDP, the weighting factor is based solely on likelihood of a particular "state" transition with respect to the perturbation. This is, of course, the likelihood of the sampled observation given the belief.

Although Algorithm 2.2 takes an approximate hyperbelief as input, it can also be used on a true (continuous) hyperbelief. A pre-processing step needs to be used which samples the continuous hyperbelief and converts it to a set of particles. Also, a large number of variations of this algorithm are useful in practice. In particular, it is often useful to sample a number of observations for each belief in $\hat{\beta}_t$. The resample procedure can then be used to reduce the number of particles back to N so the size is fixed over time.

2.5 Summary

We have described and discussed the mathematical details of a generic system described by a POMDP in three tiers of characterization. The first level describes how the system model evolves, which is hidden from an external

observer. Treating the process model of the system as a latent variable and filtering a pdf estimate of that variable based on observations gives us the belief filtering equations. However, if observations are not available, for example when trying to evaluate system behavior in the future, the belief filtering equations become the process model of an MDP in the belief space. The possible future outcomes of this process can be explored probabilistically by looking at the posterior probability on the belief space conditioned on the initial belief and the policy. This is referred to as the hyperbelief evolution of the system, and is deterministic. Of course, this benefit comes with the cost of lifting the system representation into an infinite-dimensional system which, in practice, cannot be represented with a fixed number of parameters. We can employ particle filtering in the hyperbelief space, termed hyper-particle filtering, to estimate the evolution of the system.

2.6 Further Resources

There is a large volume of literature available for stochastic processes. In particular, for expository discussion of POMDPs refer to the tutorials in [9, 6]. More detailed analysis of stochastic processes can be found in [10] and the portion of [5] concerned with stochastic systems.

The literature on particle filtering is vast. An excellent survey is available in [7]. For additional details on hyper-particle filtering, consult [8].

CHAPTER 3

POMDP OPTIMIZATION

In Chapter 2, we discussed the equations governing the evolution of a POMDP. Of course, this process is controlled, meaning that we have a control input u_t that affects the dynamics of the state, belief, and hyperbelief processes at every time step. Choosing the proper controls to apply to the system in order to achieve an engineering objective is the core problem addressed by this thesis, and in this chapter we formalize this goal explicitly.

Often there is a distinction drawn between a policy and a plan, and different communities bring different preconceptions to what these terms mean. Because one cannot know the process evolution of a stochastic system for an open-loop sequence of controls, it makes sense that the optimal policy must be closed-loop, unlike the case for deterministic systems. For this reason, we use the terms *learning*, *planning*, and *optimizing a policy* interchangeably to mean computing a belief-feedback policy. Choosing the best open-loop sequence of controls to achieve some objective is not addressed in this thesis, and for the general POMDP problem will be a suboptimal approach.

This chapter begins by mathematically defining the POMDP optimization problem and the necessary accompanying concepts in Section 3.1. In Section 3.2, we discuss dynamic programming and the value function, which is a ubiquitous tool in optimization theory. This discussion is primarily a prerequisite to discussing the main classes of approach to the POMDP optimization problem in Section 3.3. This section concludes with a short discussion of the intractability of these methods for large systems. Thus, in Section 3.4 we discuss some specific optimization methods broken into a rough taxonomy of approaches.

3.1 Defining the POMDP Optimization Problem

The quality of a particular behavior of the system is quantified in terms of an objective function. An objective function, in general, essentially maps a system trajectory (and possibly some additional degrees of freedom) to a scalar that represents the goodness of that trajectory. If the range of the objective function represents goodness it is called a reward function, and optimization entails maximizing this quantity. If the range represents badness, it is called a cost function and optimization entails minimization. Either formulation can be converted to the other interchangeably.

We consider problems with a stage-wise separable, belief process sample path cost function, i.e., cost functions that can be written in the form

$$j_T(b_0, u_{0:T}, y_{1:T}) = \sum_{t=0}^{T} \gamma^t l(b_t, u_t) \tag{3.1}$$

where $\gamma \in (0, 1]$ is called a discount factor, b_0 is a pdf over the process starting state, and $y_{1:T}$ is an instantiation of the observation process. The function is said to be stage-wise separable because it can be written as a summation where each term in the sum corresponds to the behavior of a specific stage. Thus, we have a function $l : \mathcal{P}_b \times \mathcal{U} \to \mathbb{R}^+$ called the stage cost. It is important to note that, like our formulation of the process and observation models, the stage cost is time-invariant.

We choose to quantify our cost in terms of the belief process because, since the state process is latent, we have to quantify behavior of the underlying process statistically. The stage cost function is often the expected cost of a system-specific state cost function mapping $\mathcal{X} \times \mathcal{U}$ to $\mathbb{R}^+$. More general cost functions, e.g., entropy of the belief, are also useful, so we will not restrict the stage cost unnecessarily.

Equation (3.1) specifies the cost of a sample path for a fixed sequence of observations. At optimization time, the sequence of observations is still unknown and, thus, we consider the expected cost where the expectation is taken over future observations.

3.1.1 Policies and Expected Policy Cost

A *feedback policy* is a mapping from some feedback information structure to a control. We are interested in belief-feedback policies $\pi : \mathcal{P}_b \to \mathcal{U}$. Denote the space of all belief-feedback policies as Π. Given a stage-wise separable, time-invariant cost function, the optimum policy for a POMDP system belongs to the class of belief-feedback policies [5].

The T-stage sample path cost under policy π is characterized by the equation

$$j_T(b_0, \pi, y_{1:T}) = \sum_{t=0}^{T} \gamma^t l\left(b_t, \pi(b_t)\right) \tag{3.2}$$

for a fixed set of observations. The time evolution of the belief process, i.e., b_t, can be explicitly characterized using the belief transition operator and sequence of observations.

$$b_t = \phi_{y_t, \pi(b_{t-1})} \cdots \phi_{y_1, \pi(b_0)} b_0$$

The expected T-stage cost starting from b_0 is

$$J_T(b_0, \pi) = \mathrm{E}\left[j_T(b_0, \pi, \boldsymbol{y}_{1:T}) | b_0, \pi\right] \tag{3.3}$$

We use the lower-case j to represent cost along a single sample path, while the upper-case J represents the expected cost over all sample paths. The infimum of (3.3), taken over all belief feedback polices, will be (non-strictly) better than the infimum over the expected value of (3.1), taken over all sequences of controls. This is due to the presence of feedback.

In this dissertation, we use two cost structures that can be characterized by (3.3): infinite horizon discounted cost and total cost with retirement. We can also consider much more general cost models, such as average total cost or statistics other than expected value. Infinite horizon discounted cost considers an infinite number of time stages, and is typically written

$$J^{\inf}(b_0, \pi) = \lim_{T \to +\infty} J_T(b_0, \pi)$$

which will converge if l is bounded[1] and $\gamma \in (0, 1)$. Although not a hard rule, this criterion is particularly useful for systems that persist indefinitely,

[1] We assume $|l(b, u)|$ is bounded for all b and u.

or when the optimal behavior is cyclic. Using a total cost criterion implies $\gamma = 1$, so cost will grow unbounded unless eventually $l(b_t, u_t) \to 0$. The retirement condition can be thought of as the system having a special control that can be used at any stage that allows the system to pay no more cost in the future. However, there is a cost associated with using this control, dependent on the current belief state. The total cost with retirement criterion is written

$$J^{\text{ret}}(b_0, \pi) = \text{E}\left[J_{\boldsymbol{T_r}}(b_0, \pi) + s(b_{\boldsymbol{T_r}})|b_0, \pi\right] \tag{3.4}$$

where $\boldsymbol{T_r}$ denotes the dynamically chosen (by the control policy) retirement stage and $s : \mathcal{P}_b \to \mathbb{R}^+$ denotes the retirement cost. This criterion is particularly useful for systems with a goal-seeking behavior.

As an aside, nothing in the development of this dissertation restricts the use of a stage-wise separable hyperbelief cost function, i.e.,

$$J_T^{\beta}(b_0, \pi) = \sum_{t=0}^{T} \gamma^t l^{\beta}(\beta_t, \pi)$$

This could express, for example, worst-case cost or be used to choose a sequence of controls that minimizes the variance of sample path costs over many trials. However, we do not use this criterion for any of the experiments in this dissertation and have not experimented with it, so we formulate our problem in terms of belief cost and leave this as a note.

3.1.2 Problem Objective

Our objective is to learn a policy that minimizes[2] the expected cost starting from an initial belief b_0. Specifically, this is formally defined as finding the tuple (π^*, J^*) where

$$\pi^* = \arg\inf_{\pi \in \Pi} J(b_0, \pi)$$
$$J^* = \inf_{\pi \in \Pi} J(b_0, \pi)$$

[2]Many POMDP planners are concerned with maximizing reward, but again these two formulations are entirely equivalent.

and J refers either J^{inf}, J^{ret}, or some other cost structure. As mentioned previously, different communities have different terminology regarding the POMDP problem. This same objective is referred to as optimizing a POMDP, solving a POMDP, or POMDP planning.

It should be noted that in many cases one must restrict the class of policies beyond the set of all belief-feedback policies. For example, our approach is predicated on choosing the best policy from among the set of switched belief-feedback policies with pre-chosen policy modes. (We will discuss this in Section 4.1.) In these cases, the minimum (and argument of the minimum) are taken over all policies in this restricted policy set.

3.2 Dynamic Programming and the Value Function

Dynamic programming is a method for solving problems by breaking them into smaller, simpler subproblems. Since optimum trajectories for dynamic systems have optimal substructure,[3] dynamic programing can be utilized to characterize optimum POMDP trajectories. Specifically, dynamic programming simplifies the decision process by finding one-stage optimum trajectories, and then putting them together to find a full optimum trajectory. We will give a short, functional discussion of dynamic programming and the value function here, primarily to establish notation. However, a reader not already familiar with these principles could consult [11] or the more expository discussion of [5].

At the core of the dynamic programming procedure for dynamic systems is the value function. Define the value function $V : \mathcal{P}_b \to \mathbb{R}^+$ to be the minimum expected cost-to-go, i.e., the remaining cost of a trajectory that has reached point b using the optimum policy to specify controls

$$V(b) := \min_{\pi \in \Pi} \lim_{T \to +\infty} J_T(b, \pi) \tag{3.5}$$

Using the definition of the value function and the stage-wise separability of J, this can be reformulated in terms of a one-step decision problem. Thus,

[3] A problem is said to have *optimal substructure* if its optimal solution can be obtained by the combination of optimal solutions of its subproblems.

the value function satisfies the relationship

$$\begin{aligned} V(b) &= \min_{u \in \mathcal{U}} \left\{ l(b,u) + \gamma \mathrm{E}\left[V(\phi_{\boldsymbol{y},u} b) \right] \right\} \\ &= \min_{u \in \mathcal{U}} \left\{ l(b,u) + \gamma \int \mathrm{f}_{\boldsymbol{y}_t | \boldsymbol{b}_t, u_{t-1}}\left(y|b,u\right) V(\phi_{y,u} b) dy \right\} \end{aligned} \tag{3.6}$$

This is referred to as the stochastic Bellman equation, and is central to nearly all optimization methods for MDPs and POMDPs. Thus, the value function can be computed recursively by noticing that the expected cost-to-go can be expressed as the sum of the next stage cost plus the expected cost-to-go after one stage. This equation also specifies the optimal policy as

$$u^* = \pi^*(b)$$

where u^* is defined as the control that achieves the minimum at b in (3.6), and π^* is the policy that achieves the minimum in (3.5).

The value of the pdf $\mathrm{f}_{\boldsymbol{y}_t | \boldsymbol{b}_t, u_{t-1}}\left(y|b,u\right)$ can be computed using the POMDP model and the total law of probability

$$\begin{aligned} \mathrm{f}_{\boldsymbol{y}_t | \boldsymbol{b}_t, u_{t-1}}\left(y|b,u\right) &= \int \mathrm{f}_{\boldsymbol{y}_t | \boldsymbol{b}_t, u_{t-1}, \boldsymbol{x}_t}\left(y|b,u,x\right) \mathrm{f}_{\boldsymbol{x}_t | \boldsymbol{b}_t, u_{t-1}}\left(x|b,u\right) dx \\ &= \int \mathrm{f}_{\boldsymbol{y}_t | \boldsymbol{x}_t, u_{t-1}}\left(y|x,u\right) b(x) dx \end{aligned}$$

Note that by using an integral throughout this development we have assumed the observation space is continuous. A discrete observation space can be handled equivalently using a summation.

For notational convenience, we will define the *policy value function* $V^\pi : \mathcal{P}_b \to \mathbb{R}$. The policy value function maps beliefs to a value (expected cost-to-go) under a suboptimal policy.

$$\begin{aligned} V^\pi(b) &:= \lim_{T \to +\infty} J_T(b, \pi) \\ &= l(b, \pi(b)) + \gamma \int \mathrm{f}_{\boldsymbol{y}_t | \boldsymbol{b}_t, u_{t-1}}\left(y | T_{\pi(b)} b, \pi(b)\right) V^\pi(\phi_{y,\pi(b)} b) dy \end{aligned} \tag{3.7}$$

When we say the "value function under a policy" or "using a policy" we are referring to the policy value function.

If one can solve (3.6) in closed form, then the value function can be used to compute the optimal control for every belief. This is the best possible

scenario, and should be exploited if possible. However, for many systems, solving (3.6) or (3.7) in closed form is very hard, and in some cases impossible. The next section discusses some canonical approaches to find the value function and generate the optimal policy. These approaches tend to be computationally difficult for many problems of significant size, and so a number of approximation approaches have been introduced. We discuss these in Section 3.4.

3.3 Exact POMDP Optimization Techniques

We divide the discussion of canonical approaches into three main categories. Dynamic programming methods utilize the full system model and explicitly use the Bellman equation. Model-free methods, which are sometimes called Monte Carlo or reinforcement learning methods, operate based on sample paths from system simulation. Thus, they do not require a system model in order to compute an optimal solution. Finally, a linear programming formulation is a useful alternate approach, particularly in the multi-objective case and for analysis of the problem.

In this dissertation, we mainly discuss dynamic programming methods as they are the basis of our work. However, we briefly discuss the model-free and linear programming approaches to the problem as they are both conceptually important and also precursors to several related approaches.

3.3.1 Dynamic Programming Methods

Dynamic programming methods can be used to compute optimal policies given a perfect model of the process and observation models of the system. We assume we have such a model throughout this dissertation, and so our approach falls into the dynamic programming category. There is a wealth of dynamic programming theory and study, and an excellent, detailed discussion of these approaches can be found in [5], and in many other sources.

Dynamic programming methods explicitly use the Bellman equation to perform a *backup* to iteratively estimate the value function. Although different backups are possible, the core idea is to "back up" the value from the set of (nonzero likelihood) successor states to produce the value at the

predecessor state.

The value iteration algorithm, which is specified by

$$\hat{V}^{i+1}(b) = \min_{u \in \mathcal{U}} \left\{ l(b,u) + \gamma \int \mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{b}_t,u_{t-1}}(y|T_u b, u)\, \hat{V}^i(\phi_{y,u} b) dy \right\}$$

is a dynamic programming procedure that utilizes the backup operation. Assuming we have an estimate of the value function, denoted[4] $\hat{V}^i : \mathcal{P}_b \to \mathbb{R}$, we use this equation to perform a sequence of backups to generate a new estimate $\hat{V}^{i+1}$. Under some fairly general conditions related to existence of V, it is known that $\hat{V}^i$ will converge point-wise to V. Once this is achieved or we get sufficiently close,[5] an optimal policy can be synthesized by choosing

$$\pi^*(b) = \arg\min_{u \in \mathcal{U}} \left\{ l(b,u) + \gamma \int \mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{b}_t,u_{t-1}}(y|T_u b, u)\, \hat{V}^\infty(\phi_{y,u} b) dy \right\}$$

Since the belief space is continuous, for our application the value iteration procedure will typically need to be combined with a computation of the reachable belief space or a parameterization of the value function.

The policy iteration algorithm employs a similar mechanism to iteratively update a policy. Recall that V^π, which is defined in (3.7), denotes the value under policy π. Suppose we have V^{π^i}, and for some belief we would like to know if there is a control that will produce a better expected result than the current prescribed control. Define the Q-function under π to be

$$Q^\pi(b,u) = l(b,u) + \gamma \int \mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{b}_t,u_{t-1}}(y|T_u b, u)\, \hat{V}^\pi(\phi_{y,u} b) dy$$

which essentially gives the expected cost of using control u at b and then following policy π in the future. Assuming we have a policy π^i, a single stage of policy iteration is characterized by improving to π^{i+1} by choosing

$$\pi^{i+1}(b) = \arg\min_{u \in \mathcal{U}} Q^{\pi^i}(b,u)$$

and then computing $V^{\pi^{i+1}}$. Under the appropriate conditions, policy iterates converge to the optimal policy. Again, a reachability computation or parameterization will typically be required in practice.

[4] We have used a subscript to indicate the time step of the process, and now we use a superscript to indicate the iteration of the optimization algorithm.

[5] In some cases convergence is only guaranteed in the limit.

Dynamic programming approaches can be thought of as a combination of policy evaluation and improvement steps. For example, in policy iteration we have a full policy evaluation step and then one increment of policy improvement, i.e., we backup one time. In contrast, value iteration uses a truncated policy evaluation, i.e., just one backup, and then improves the policy based on that estimate. There are many possibilities for how one can interleave evaluation and improvement. When combined with an ordering of which beliefs and controls to explore, there are a large number of algorithms that can limit the number of times beliefs and controls must be touched during execution of the algorithm by performing backups asynchronously. Some examples are depth-first search, bread-first search, A* search, and Dijkstra's algorithm [12].

3.3.2 Model-Free Learning

Often, an MDP model of system behavior is not available,[6] and one must select from a class of methods that do not explicitly utilize a model. Thus, these methods are referred to as "model-free" and can be used online, i.e., directly connected to the system as it evolves or is simulated. Because of the online nature of the methods, they can be used to solve the reinforcement learning problem [13]. As such, these methods are often referred to as reinforcement learning methods as that is where they have historically been most often encountered. Although we assume a system model and our proposed method is not model-free, we discuss these methods briefly because they are frequently used in conjunction with stochastic systems in the robotics community. Excellent surveys of reinforcement learning methods are available in [13] and [14], and a view of Q and TD learning from a control theory perspective can be found in [15].

One of the most commonly used reinforcement learning algorithms in the robotics community is Q learning [16]. We previously defined the Q function with respect to a policy, but the optimal Q function can also be defined

[6] In our case, this would result from not having a process and observation model, and thus we would not be able to write the belief transition operator.

recursively (as we did with the value function).

$$Q(b,u) = l(b,u) + \gamma \int \mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{b}_t,u_{t-1}}(y|T_u b,u) \min_{u' \in \mathcal{U}} \Big[Q(\phi_{y,u} b, u') \Big] dy$$

Notice that this is equivalent to using the value function in the integrand as

$$V(b) = \min_{u \in \mathcal{U}} Q(b,u)$$

for all $b \in \mathcal{P}_b$, but advantageous because it represents "value" for a control-belief tuple. This is advantageous because we can employ an iterative algorithm to learn the Q function, based only on sample path behavior of the process and without requiring a model of the system. The update law for the one-step Q-learning algorithm is

$$\hat{Q}^{i+1}(b,u) = (1-\zeta_i)\hat{Q}^i(b,u) + \zeta_i \left[r + \gamma \min_{u' \in \mathcal{U}} \hat{Q}^i(b',u') \right]$$

where $b' = \phi_{y,u} b$ and $r = l(b',u)$ are learned by a trial on the system, not from a system model, and

$$\zeta_i = 1/(1 + \mathrm{visits}_i(b,u))$$

is a learning rate based on $\mathrm{visits}_i(b,u)$, the number of times u has been used at b. Convergence (with probability one) of the Q-function iterates has been shown in [16], under the condition that each belief-control pair is tried infinitely often.

While we have discussed the update law for the Q function, we have not specified how belief-control pairs should be chosen, other than that each pair must be tried infinitely often. The short answer is that b is dictated by the system and u should be sampled. Thus, Q learning is referred to as an off-policy method as it learns the Q function, but does not always follow the policy that is being computed. Rephrasing, the Q learning algorithm learns the value function using suboptimal control inputs.

The most similar class of on-policy methods, i.e., methods that use the policy being learned to control the system during learning, are called Actor-Critic methods [13]. These methods have the advantage of being able to more compactly store policies and select controls and can be used to explicitly learn

a stochastic policy, i.e., a policy from the class of randomized strategies.

Recently in [17], it was shown that both Q and TD learning have strong connections to optimal control theory. These connections and new perspective on reinforcement learning algorithms may lead to new and more efficient learning algorithms that combine the advantages of dynamic programming and reinforcement learning approaches.

3.3.3 Linear Programming Approach

The linear programming approach is an interesting alternative perspective to dynamic programming. Aside from suggesting algorithms based on linear programming, rather than dynamic programming, it can handle multi-objective or constrained problems. This is because the policies learned can be drawn from the class of randomized strategies. The essence of the approach is to choose the decision variables to represent a mixed strategy, let the linear program constraints encode the belief MDP, and encode the cost function of the belief MDP in the cost function of the linear program. If the problem is unconstrained and has a single stage-wise separable cost function, the optimum policy will correspond to an extreme point of the feasible region of the linear program which corresponds to a deterministic policy. This approach for discrete states and controls and a discounted infinite horizon cost criterion is discussed in [18], and is surveyed in [15].

3.3.4 Computational Expense

One of the primary difficulties in applying dynamic programming (or other) approaches to POMDPs, viewed as MDPs in the belief space, is that the belief space is continuous. This implies either we must compute the value point-wise at every reachable belief, or find a parameterization of the value function that can be updated under the backup operation. Thus, it is not immediately clear if the value function will be representable with a finite description or computable under the value iteration or other dynamic programming procedure. In [2] this issue was resolved. Specifically, [2] showed the i^{th} value function (obtained after a finite number of backups) is finite and piecewise linear. Furthermore, [2] gave an algorithm to compute the value

function based on updating a set of linear functions, termed "α-vectors."

The most significant bottleneck in these procedures is that the complexity of the piecewise linear function, i.e., the number of pieces, grows exponentially under backups. Of course, typically all the α-vectors are not required and pruning methods can be used to minimize the overhead. Different pruning techniques were proposed in [19, 20, 21], but the computational expense still tends to be great. This is because the task of identifying all linear functions that cannot be pruned is computationally intractable as well [22]. Thus, not only does one have to potentially keep an exponentially expanding number of useful linear functions, but identifying the right functions to keep is also extremely hard. In practice, dimensionality and precision problems render most realistic POMDPs intractable using this exact approach [6, 1].

These same representational and computational issues construct similar roadblocks for model-free approaches. Thus, researchers have thus focused on finding efficient approximation methods to solve POMDP models. The next section highlights a number of key approximation techniques that can be used to solve POMDP systems.

3.4 Approximate POMDP Optimization Techniques

Due to the difficulties of finding an exact optimal policy, researchers have focused on finding approximations to the optimal policy. In this section, we provide a rough taxonomy of methods and discuss specific approaches from the literature. We emphasize that the divisions are largely meant to be illustrative by reducing emphasis on the many differentiating nuances and focusing on the most important features of the approaches. Other divisions of these techniques are certainly possible.

3.4.1 Finite Planning Horizon Methods

Since the representational complexity of the value function (or the number of sample paths one must consider) grows exponentially with planning horizon, i.e., number of backups, a prudent approximation is to limit the number of stages the planning algorithm looks forward during consideration of optimality. The extreme case of this is the greedy policy, which is the controller that

chooses

$$\pi^{g}(b) = \arg\min_{u \in \mathcal{U}} \Big[l(b, u) \Big]$$

for every belief. Unless the gradient of the one-stage cost function points in the direction of gradient of V, this policy is likely to be suboptimal. However, with a long enough planning horizon, the approximate value function can begin to approximate the true value function (and in some cases fully capture it).

This type of approach is known as a finite-memory method because it bases the learned policies on a truncated sequence of past controls and observations, i.e., an information state with finite memory. The value function for infinite horizon, discounted cost problems can be approximated with arbitrary precision by a piece-wise linear value function by computing the finite horizon $\hat{V}$ with sufficiently long horizon [23]. Other POMDP-specific finite memory approaches are [24, 25, 26].

An important consideration is the "starting" value function that is considered ground truth at the end of the planning horizon. The QMDP method [27] combines MDP and POMDP value function backups to produce an approximate value function for the POMDP. The algorithm first converts the POMDP to a fully observable MDP and performs value iteration on the state space. This value function can then be projected onto the belief space using expectation, and the value function on the belief space can be backed up an additional number of stages using standard methods. This procedure typically results in a much better approximation, for a limited number of backups, than backup on the belief space alone.

The key consideration for these methods is horizon length. For models with a useful gradient on the one-stage cost function or models that require only a short planning horizon, these methods can work exceptionally well. Unfortunately, in many applications that we consider, the planning horizon required is long. Thus, any horizon sufficiently long for good planning is computationally very expensive, and any horizon not too computationally expensive is not sufficient for good planning.

3.4.2 Monte Carlo Point-Based Methods

Recently, researchers have turned to point-based approximations of the value function employing sampling. These approaches are called Monte Carlo not only because they employ randomized sampling, but also because of their connection to reinforcement learning methods. Specifically, they operate on sample paths and not on full backups. This approach is similar to finite horizon planning in the sense that resources are allocated (probabilistically) to compute the effects of controls in the near-term more accurately than the effect many stages into the future.

Efficient approximations have been developed by randomized sampling in the reachable belief space and performing standard value iteration over the reached set of beliefs. Significant approaches in this genre (chronologically) are MC-POMDP [28], HSVI2 [29], PBVI [30], PERSEUS [31], and SAR-SOP [32]. Aside from MC-POMDP, these methods are designed for discrete POMDP systems. The PBVI algorithm has been extended to operate on continuous spaces in [3].

Since these approaches are point-based and operate in only the reachable belief space, they are best used as online methods assuming the control loop is slow enough for a quality solution to be generated. A survey of these methods, as well as other similar online approaches, can be found in [33]. The fundamental bottleneck of these methods is that on large, complicated systems that require a long planning horizon, it is difficult to generate a solution that looks forward a sufficiently long horizon quickly enough to keep up with evolution of the system. In contrast, if these methods are used to pre-compute an approximation of the value function of the reachable belief space it can be difficult to store and represent the solution, assuming one has a long enough pre-compute time to even generate it.

3.4.3 Projection Methods

Significant complexity challenges are posed by dimensionality of the optimization problem. These difficulties can be partially alleviated if the belief can be projected onto a lower-dimensional space that still retains sufficient information to generate a quality solution.

The AMDP method [34], and similarly the method of [35], perform value

iteration on an augmented state space. Essentially, beliefs are projected onto a lower dimensional parameter space. The value function is then approximated on this parameter space rather than in the belief space. This technique can be employed with varying levels of success, depending on how much useful information is retained by the projected beliefs. Although initially the projection transformation was often chosen ad hoc, recently researchers have explored more sophisticated belief parameterizations and projection methods, and analyzed the quality of those projections explicitly. For example, in [36] a continuous belief is projected, using particle filtering methods, onto the exponential family of densities.

A special case of projection methods are finite-grid methods. Essentially, the value function is approximated using a deterministic sampling strategy, i.e., a finite grid. Thus, some finite-grid approaches that approximate the belief space by a grid of points are [37, 38, 39]. By backing up the value only over these grid points, a tractable number of operations is ensured.

The fundamental difficulty with projection methods is ensuring the projection operator preserves the information necessary to generate a quality policy. Thus, the analysis that guarantees this must be done model by model. If such a projection does exist, it makes good engineering sense to utilize it. However, often the dimensionality of the maximally reduced belief representation (that still yields a quality policy) still has sufficiently high complexity to cause severe tractability problems.

3.4.4 Compression Methods

Compression methods attempt to use model reduction to simplify the complexity of the optimization process by simplifying the POMDP model. The simplification is done in such a way that the belief transition operator maintains its qualitative behavior but is significantly simpler to represent and compute.

Typically in the context of POMDPs this has been done for discrete state systems using techniques like principle component analysis on the belief process and observation operators or related quantities, e.g., [40, 41, 42]. The core approach of these models is to build a policy for an approximate, coarse system model and then apply it to the original system.

Again, if this model reduction can be used without significantly degrading performance, it is a prudent engineering decision to employ it. However, for many large, complicated systems a (prudent amount of) model reduction does not yield sufficient complexity reductions to circumvent computation problem in practice.

3.4.5 Roadmap-Style Methods

The roadmap-style of approach was discussed at length in Chapter 1, as our approach falls into this category. The main goal of a roadmap approach is to leave the local aspects of control, i.e., specific controls at specific stages, to a local planner and consider the coarse evolution of the system.

These methods have significant connection to hierarchical planning methods. Hierarchical POMDP methods have been explored in the literature. Planning with a pre-defined hierarchy of tasks has been explored in [43] and [44]. Other methods, such as [45, 46, 47], attempt to discover a hierarchy of tasks to use for planning.

Other methods that can be categorized as purely roadmap-style use sampling to select target locations and construct a graph between these targets. Such methods include the Belief Roadmap (BRM) in [48], the Stochastic Motion Roadmap (SRM) in [49], and the Sampling Hyperbelief Optimization Technique (SHOT) in [50]. The BRM and SRM algorithms sample points in the belief space and attempt to connect those points using a local planner. Uncertainty is characterized in the BRM using an extended Kalman filter (EKF), and based on that approximation, the mean of the uncertainty is assumed controllable in a parameterized uncertainty space. They then use a local planner to reach sampled values of the mean parameter and rate the quality of paths based on the variance parameter of the EKF. The SRM algorithm uses sample path simulation to attempt to discover the likelihood of two configurations being connected and searches only for feasible paths. The SHOT algorithm attempts to build a graph in the hyperbelief space.

Switched approaches to solving uncertain systems are not new. Approaches for MDP's such as [51] and [52], like roadmap methods, are hierarchical. Semi-Markov decision processes like [52] use macro-actions or options, feedback or open-loop policies that persist until a termination condition is met, to

temporally abstract the problem to reduce the number of controls considered. These methods do not consider partially observable processes.

Our approach belongs to a recently emerging class of methods, e.g., [53, 54, 55], that use *local policies* or *macro-actions* to construct a POMDP policy. Essentially, rather than optimizing single controls (single time steps), we optimize the choice of controls for a short time span (over multiple time steps).

The fundamental difficulty of roadmap-style methods is ensuring the connection strategies are sufficient to generate a quality policy. This problem is hard to quantify, and we attempt to circumvent it by choosing an approach based on domain knowledge. Significant future exploration will be required to definitively study the quality of these approaches.

3.5 Summary

In this chapter, we defined the POMDP optimization problem and the necessary accompanying concepts. We discussed dynamic programming and the value function, which is a ubiquitous tool in optimization theory. This discussion was a prerequisite to discussing the main classes of approach to the POMDP optimization problem. Since these methods are intractable, severe computational hurdles prevent them from being used for many large, complicated systems. We discussed some specific optimization methods broken into a rough taxonomy of approaches. In the next chapter, we present our optimization approach based on sampling-based learning of switched policies.

CHAPTER 4

SAMPLING-BASED LEARNING OF SWITCHED POLICIES FOR POMDPS

In this chapter, we present a sampling-based algorithm that incrementally computes a switched belief-feedback policy for a POMDP. The utility of our approach is predicated on the availability of a pre-defined set of local policies, which are assumed to be provided to the algorithm. At a conceptual level, a local policy is similar to the local planners used by sampling-based motion planning algorithms (see [56]); i.e., the sampling-based planner explores the configuration space of the process by relying on the local planner to handle the details of the connection strategy between sample points. Local policies are termed "local" because they are meant to be used for a small number of consecutive stages (locally in the belief space), not because they are defined over a restricted domain. In fact, a local policy consists of a belief-feedback policy (defined over the entire belief space) and stopping condition.

The role of the optimization algorithm is to synthesize a set of local policies into a single policy that is effective (with respect to minimizing the objective function) over the reachable belief space for the POMDP, given a specific initial condition. The policy generated by the algorithm belongs to a class of policies we refer to as *switched policies*, which is formally defined in Section 4.1. This restriction on the policy space provides a natural framework for temporal abstraction. Utilizing this framework, we can learn policies that encode strategies that can only be discovered when evaluating a long time horizon, and can do so in significantly less time than algorithms that learn policies one control at a time. Taking into account limited computing resources, this is often the key to finding quality solutions for very large POMDPs.

An important caveat is, for this method to perform well in practice, we must have some prior knowledge in the form of local policies. Local policies can sometimes be computed automatically by methods in the literature, e.g., [45, 46, 47], but this approach is not always optimal for planning as these

methods can be, in some cases, as difficult as learning an optimal policy. We suggest that our proposed method is most useful when it can be used on a problem where some domain or expert knowledge can be specified by an external process, e.g., an engineer or system designer. This paradigm is particularly relevant to many applications we are concerned with, specifically, cases where one can quickly determine locally optimal policies or, at minimum, strategies that may be useful for a few stages. The role of our proposed algorithm is to find the best way to compose these local policies, a task that is, conversely, typically difficult for an engineer or system designer.

In this chapter, we begin by formally describing the class of switched policies in Section 4.1. We then discuss the evolution of a POMDP under a local policy, which we term an *expansion* and explicitly define in Section 4.3. This computation is useful because it characterizes the evolution of the system from a point in the belief space until the next switching time. Unfortunately, the expansion operator is computationally expensive, so we discuss approximations in Section 4.3.2. Once these tools are in place, we use them to construct an algorithm to learn a switched policy in Section 4.4. Because this algorithm is computationally impractical, we conclude by presenting a sampling-based approximation algorithm in Section 4.4.2, which is one of our main results. We conclude with a discussion of our algorithm in Section 4.5.

4.1 Switched Policies

A switched[1] policy is a policy that chooses a control based on the active *mode* of the switched policy and the current belief of the system. It can be used to combine a set of local policies into a composite policy. A switched policy can be used in place of a standard belief-feedback policy; i.e., given a belief, a switched policy returns a control. However, a switched policy maintains internal state (the active local policy and the number of stages elapsed since the last switch), so implementation of the policy will differ from a standard belief-feedback policy.

A local policy[2] ψ is a two-tuple composed of a (possibly time-varying)

[1]For additional discussion of switched and hybrid systems, see [57].

[2]Similar concepts are sometimes called *options* [52] or *macro-actions* [54]. We use the terminology *local policy* to make clear that our approach is to use feedback policies, not open loop sequences of controls.

belief-feedback policy and a stopping condition, i.e.,

$$\psi = \{\pi_\psi, a_\psi\}$$

where the stopping condition

$$a_\psi : \mathcal{P}_b \times \mathbb{Z}^+ \to \{0, 1\}$$

indicates when the planner should consider changing local policies based on the current belief and the number of stages since the last switch. Switching conditions may depend on the number of stages since the last switch, but not the absolute stage number. A local policy encodes a mode plus stopping condition of the switched policy. We denote the set of available local policies Ψ. Typically, Ψ will either be finite or finitely parameterizable with a continuum of values on at least one of the parameters. Additional constraints may be imposed on this subset of the policy space. For example, it is useful to restrict the allowable stopping conditions such that each local policy will stop in finite time with probability one.

We denote by $\overline{\pi}$ a switched policy, which consists of a set of local policies Ψ and a switching law

$$\alpha : \mathcal{P}_b \to \Psi$$

A switched policy contains a state component indicating the current (active) local policy and the number of stages elapsed since the most recent switch. Thus,

$$\overline{\pi} : \mathcal{P}_b \times \Psi \times \mathbb{Z}^+ \to \mathcal{U}$$

If the active local policy is ψ, then the control generated by the switched policy satisfies

$$\overline{\pi}(b, \psi, \tau) = \pi_\psi(b)$$

If the stopping condition is met, i.e., $a(b, \tau) = 1$, then α chooses a new local policy from Ψ to become active. Thus, α does not specify a new local policy at every time step, but instead only when a local policy terminates.

To summarize, a switched policy $\overline{\pi}$ is a mechanism for combining a set of local policies. Each local policy ψ, when activated, has a belief-feedback policy π_ψ that specifies controls for the POMDP. The stopping condition of the specific active local policy a_ψ acts as a guard that only allows the

active local policy to change when the stopping condition is met. Once the stopping condition is met, the switching law α chooses the next local policy to use from Ψ.

4.2 Policy Value Function for Switched Policies

In this section, we write a version of the stochastic Bellman equation to compute the policy value function for belief-feedback policies, i.e., (3.7), suitable for computation of a policy value function for a switched policy. Recall that a switched policy has state, specifically (ψ, τ) where ψ is the active mode (specified as a local policy) and τ is the number of stages since the last switch. Define the policy value function to be the expected cost-to-go if the POMDP and switched policy have combined state (b, ψ, τ). An analog to (3.7) is then

$$\begin{aligned} V^{\overline{\pi}}(b,\psi,\tau) &= l(b,\pi_\psi(b)) \qquad (4.1)\\ &\quad + \mathrm{E}\left[a_\psi(\boldsymbol{b'},\tau{+}1)\gamma V^{\overline{\pi}}(\boldsymbol{b'},\alpha(b'),0) + (1{-}a_\psi(\boldsymbol{b'},\tau{+}1))\gamma V^{\overline{\pi}}(\boldsymbol{b'},\psi,\tau{+}1)\right] \end{aligned}$$

where $\boldsymbol{b'} = \phi_{\boldsymbol{y},\pi_\psi(b))}b$. The expression inside the expectation is basically selecting one of two possible alternatives for the next state of the system. Either the system switches at $(\boldsymbol{b'}, \psi, \tau + 1)$, in which case τ is reset to zero and a new policy is chosen based on the switching law, or no switch occurs.

We can recursively replace values where $\tau > 0$ on the right-hand side of (4.1). The recursion will have finite depth presuming the active local policy is guaranteed to stop in finite time. The resulting equation is

$$V^{\overline{\pi}}(b,\psi,\tau) = \mathrm{E}\left[j_{\bar{\boldsymbol{t}}}(b,\pi_\psi,\boldsymbol{y}_{1:\bar{\boldsymbol{t}}}) + \gamma^{\bar{\boldsymbol{t}}}V^{\overline{\pi}}(\boldsymbol{b}_{\bar{\boldsymbol{t}}},\alpha(\boldsymbol{b}_{\bar{\boldsymbol{t}}}),0)\right]$$

where $\bar{\boldsymbol{t}}$ denotes the next switching time for a particular sample path, which of course is a random variable due to the probabilistic belief process evolution.

Since we are only interested in backing up the value to switching points, i.e., points where $\tau = 0$, we can suppress the dependence on the active local policy (as it will be specified by the switching function and the current belief) and τ. Thus, for every b where a switch occurs and assuming a switch has

just occurred

$$V^{\overline{\pi}}(b) = \mathrm{E}\left[j_{\bar{\boldsymbol{t}}}(b, \pi_{\alpha(b)}, \boldsymbol{y}_{1:\bar{\boldsymbol{t}}}) + \gamma^{\bar{t}} V^{\overline{\pi}}(\boldsymbol{b}_{\bar{\boldsymbol{t}}})\right]$$
$$= J_{\alpha(b)}(b, \pi_{\alpha(b)}) + \int_{\mathcal{P}_b \times \mathbb{Z}^+} \beta_{\alpha(b)}(b', \bar{t}) \gamma^{\bar{t}} V^{\overline{\pi}}(b') \; db' \; d\bar{t} \tag{4.2}$$

where the expectation is taken over $\bar{\boldsymbol{t}}$, $\boldsymbol{y}_{1:\bar{\boldsymbol{t}}}$, and $\boldsymbol{b}_{\bar{\boldsymbol{t}}}$ conditioned on b and $\overline{\pi}$. The notation J_ψ indicates the expected cost until switch. It is analogous to J_t except that the time horizon over which stage costs are summed is not fixed along all sample paths, and depends on the stopping condition of the local policy. The notation β_ψ is a probability function over stopping belief and time. Both quantities are conditioned on the starting belief and policy.

Essentially we have split the equation into an expected cost until switch and expected cost-to-go, appropriately discounted, based on the location at which the belief will reside after the switch. The unfamiliar aspect of (4.2) is due to the fact that, for the purpose of computing the discount, we need to compute a reachable set over $\mathcal{P}_b \times \mathbb{Z}^+$. If γ is one, this reduces to

$$V^{\overline{\pi}}(b) = J_{\alpha(b)}(b, \pi_{\alpha(b)}) + \int_{\mathcal{P}_b} \beta_{\alpha(b)}(b') V^{\overline{\pi}}(b') \; db' \tag{4.3}$$

which is simply the value backup over a multi-stage execution. For non-discounted systems, this simplification will typically be useful in practice as we no longer have to build additional bookkeeping to track $\bar{\boldsymbol{t}}$. Similarly, if the local policy executes for a fixed number of stages and the discount is nonzero, the factor $\gamma^{\bar{t}}$ is the only necessary addition to (4.3). For expository purposes, we will write (4.2) and (4.3) with the shorthand notation

$$V^{\overline{\pi}}(b) = J_{\alpha(b)}(b, \pi_{\alpha(b)}) + M_{b,\alpha(b)} V^{\overline{\pi}} \tag{4.4}$$

where $M_{b,\psi}$ is an operator that computes the appropriate expected discounted value at the time of switch. Specifically, given a function $V : \mathcal{P}_b \to \mathbb{R}^+$ and switching law α we have

$$M_{b,\psi} V = \int_{\mathcal{P}_b \times \mathbb{Z}^+} \beta_\psi(b', \bar{t}) \gamma^{\bar{t}} V(b') \; db' \; d\bar{t}$$

where β_ψ is conditioned on starting from belief b.

Equation (4.4) is the stochastic Bellman equation, but the stage cost and

expected cost-to-go terms are atypical. Thus, we have converted the canonical form to a form that allows each backup to occur over one local policy expansion. Thus, our gains in performance come firstly from reducing the complexity of the search graph over which we optimize by considering policy decisions at a coarser level, but also from being able to utilize approximation methods that are standard in the robotics community, e.g., particle filters. If we can compute J_ψ and β_ψ, we can optimize the system's evolution with respect to local policies rather than individual controls. These computations are the subject of the next section.

4.3 POMDP Evolution under a Local Policy

In order to perform value backups, for example for (4.4), we must be able to compute the expected result of applying a local policy at a belief. Specifically, we need to compute the expected cost accrued before the local policy switches and the probability distribution over the set of belief point and time combinations where the switch will occur. Comparing to the canonical dynamic programming approach to optimize a control at a single time step, these parameters are analogous to the one-stage cost and the one-stage transition probabilities. We refer to a mapping from a belief and local policy to these parameters as an *expansion*. Thus, an expansion ϑ is the transformation

$$\vartheta : \mathcal{P}_b \times \Psi \to \mathbb{R}^+ \times \mathcal{P}_{\beta t}$$

where $\mathcal{P}_{\beta t}$ is an augmented hyperbelief space, and refers to the set of probability functions over the space $\mathcal{P}_b \times \mathbb{Z}^+$. The $\mathbb{R}^+$ in the range refers to the expected cost until switch. Note that $\beta_\psi \in \mathcal{P}_{\beta t}$ is a pdf over switching time and location, not a set of time-indexed hyperbeliefs. An expansion parametrizes the effect of applying a local policy at a specific belief. The remainder of this section is devoted to computing the expansion operator and approximating that computation.

4.3.1 The Expansion Transformation

Evaluating an expansion is a fairly straightforward procedure that is analogous to simulation of the system in the hyperbelief space, with additional bookkeeping wrapped around the hyperbelief transition function. The additional effort is due to the switching time being a random variable, and not known a priori. Let b_0 be the belief from which the local policy is started. Recall J_ψ is the expected cost until switch, and define β_t^c to be the hyperbelief at stage t that accounts for trajectories in the belief space that have not switched. Finally, define $\beta_\psi \in \mathcal{P}_{\beta t}$ to be the augmented hyperbelief that maps switching time and belief to likelihood[3] of switch occurring at that point (belief and time).

Using β_t^c, we compute the probability that a switch will occur at time $t+1$ conditioned on the event that a switch has not occurred at a previous stage during this expansion.

$$\xi_{t+1} := \mathrm{p}\left(a_\psi(\boldsymbol{b}_{t+1}, t+1) = 1 | \beta_t^c\right) = \int_{\mathcal{P}_b} \mathbb{I}[a_\psi(b, t+1) = 1] \cdot (\Phi_{\pi_\psi} \beta_t^c)(b)\ db \tag{4.5}$$

The indicator function acts as a filter to select all probability density of the hyperbelief where the stopping condition a_ψ is one. Let the probability that the POMDP will have switched at or before stage t be

$$p_t^{\mathrm{s}} = \mathrm{P}\left[a_\psi(\boldsymbol{b}_\tau, \tau) = 1 \text{ for } \tau \leq t\right]$$

We compute this quantity recursively using (4.5)

$$p_{t+1}^{\mathrm{s}} = p_t^{\mathrm{s}} + (1 - p_t^{\mathrm{s}})\xi_{t+1}$$

This equation can be derived from the total law of probability when conditioning on the event that the process has already switched.

We filter β_t^c starting from $\beta_0^c = \beta_0 = \delta(b - b_0)$ using

$$\beta_{t+1}^c(b) = \frac{1}{1 - \xi_{t+1}}\ \mathbb{I}[a_\psi(b, t+1) = 0] \cdot (\Phi_{\pi_\psi} \beta_t^c)(b)$$

[3]The uncertainty captured by this pdf is due to uncertainty of which switching point will be reached first. Once one of the switching points is reached, a switch will occur surely.

for all $b \in \mathcal{P}_b$. This equation is based on the hyperbelief transition operator, which is the process model for the POMDP description in the hyperbelief space. The standard operator is modified to remove support from the pdf at all beliefs that will trigger the switching condition at this particular time stage. Of course, if probability is removed, this will result in a functional that is not a pdf, so the expression must be normalized by taking into account the probability that the process switches at this stage.

At this point, we have discussed a sequence $\{\beta_t^c, p_t^{\mathrm{s}}\}$ that computes the probability that there is a switch by time t, and the hyperbelief for trajectories that have not yet switched. We can use this sequence to incrementally compute the expected cost until switch and the pdf of where the switch occurs. Let J_t^ψ denote the expected cost of the POMDP accrued between stages 0 and t if a switch has not occurred. It can be computed using the difference equation

$$J_{t+1}^\psi = J_t^\psi + (1 - p_t^{\mathrm{s}})\, \mathrm{E}_{\boldsymbol{b}_{t+1}|\beta_{t+1}^c} \left[\gamma^{t+1}\, l(\boldsymbol{b}_{t+1}, \pi_\psi(\boldsymbol{b}_{t+1})) \mid \beta_{t+1}^c\right]$$

which adds expected cost at stage $t+1$ to the expected cost accrued through stage t. The pdf over the switching point, defined on the space $\mathcal{P}_b \times \mathbb{Z}^+$, can be incrementally constructed as t increases with the first t elements of the sequence $\{\beta_t^c, p_t^{\mathrm{s}}\}$, i.e.,

$$\beta_\psi(b, t) = (1 - p_{t-1}^{\mathrm{s}}) \cdot \mathbb{I}[a_\psi(b, t) = 1] \cdot \beta_t^c(b)$$

for $b \in \mathcal{P}_b$.

This construction procedure terminates when $p_t^{\mathrm{s}} = 1$, as a switch will have occurred on all trajectories. At this point, no additional cost will be accrued before switching (so $J_t^\psi = J_\psi$) and the switching pdf construction will be complete, so the computation of the expansion operator is complete. However, there is no explicit guarantee p_t^{s} will equal one in finite time, and examples can easily be constructed where an infinite number of steps is required to guarantee a switch will occur, e.g., any event that can be shown to occur almost surely, but not surely. In practice, for this framework to be useful, local policies should be designed to enforce a finite stopping time for all trajectories.

4.3.2 Approximating the Expansion Transformation

Unfortunately, the exact procedure discussed in Section 4.3.1 cannot typically be achieved in practice. Simulating the system in the hyperbelief space requires us to account for all system trajectories. For example, in the discrete POMDP case, this means accounting for $O(|\mathcal{Y}|^t)$ trajectories in the belief space, where t is sufficiently large to guarantee $p_t^{\mathrm{s}} = 1$. As the number of stages for which the local policy executes increases, the number of resources needed exponentially increases.

We propose an algorithm to approximate ϑ using a sampling approach. This core tool for approximating J_ψ and β_ψ is based on a set of Monte Carlo experiments. An approximation based on a set of Monte Carlo random walks may be particularly useful if the approximation will be computed (and refined) incrementally, rather than in a single block of computation. A more elegant solution for the case of a fixed amount of computation per expansion is based on hyper-particle filtering, which refers to particle filtering on the belief space and was discussed in Section 2.4. Hyper-particle filtering can enforce some dissimilarity between the sample paths simulated, which can result in a better approximation using a similar amount of computation resources to a random walk. For this reason, we typically choose an approach based on hyper-particle filtering.

Although our algorithm is similar in principle to the one presented in Section 2.4, we must incorporate additional bookkeeping to account for the switching time being a random variable. This is completely analogous to the method of Section 4.3.1, where we augmented the standard hyperbelief filtering equation to compute the expansion. The difference between the method of this section and Section 4.3.1 is that a particle filter approximation of the pdf β_t^c will stand in for the exact pdf.

Algorithm 4.1 specifies our approximation algorithm for ϑ explicitly. The algorithm essentially constructs and iterates approximations of the sequences discussed in Section 4.3.1, by approximating β_t^c by constraining it to the parametrization given in (2.7). Because we cannot evaluate all sample paths, the function SampleObservations is introduced that chooses observations by Monte Carlo experiments. Although many sampling algorithms have been explored for particle filtering, a good estimate can often be obtained for POMDP systems by sampling a sufficiently large number of observations

Input: b_0, ψ
Output: $\hat{\beta}_\psi$, $\hat{J}_\psi$

$\hat{J}_\psi \leftarrow 0,\ \hat{\beta}_\psi \leftarrow \emptyset$
$t \leftarrow 0,\ \hat{p}^{\mathrm{s}}_t \leftarrow 0$
$\hat{\beta}^c_0 \leftarrow \{(b_0, 1)\}$
while $\hat{p}^{\mathrm{s}}_t < 1$ **do**
  $\hat{\beta}^c_{t+1} \leftarrow \emptyset$
  $\hat{c} \leftarrow 0$
  $\hat{\xi}_{t+1} \leftarrow 0$
  foreach $(b^{(i)}_t, w^{(i)}_t) \in \hat{\beta}^c_t$ **do**
    $u_t \leftarrow \pi_\psi(b^{(i)}_t)$
    $\hat{c} \leftarrow \hat{c} + l(b^{(i)}_t, u_t) w^{(i)}_t$
    $Y \leftarrow$ SampleObservations$(u_t, b^{(i)}_t, N_{\mathrm{obs}})$
    foreach $y_t \in Y$ **do**
      $b_{t+1} \leftarrow \phi_{y_t, u_t} b^{(i)}_t$
      $p_y \leftarrow \mathbf{1}' O_{y,u} T_u b_t$
      if $a_\psi(b_{t+1}, t+1) = 1$ **then**
        $\hat{\beta}_\psi \leftarrow \hat{\beta}_\psi \cup \{(b_{t+1}, t+1, (1 - \hat{p}^s_t) p_y w^{(i)}_t)\}$
        $\hat{\xi}_{t+1} \leftarrow \hat{\xi}_{t+1} + p_y w^{(i)}_t$
      end
      else
        $\hat{\beta}^c_{t+1} \leftarrow \hat{\beta}^c_{t+1} \cup \{(b_t, p_y w^{(i)}_t)\}$
      end
    end
  end
  $\hat{J}_\psi \leftarrow \hat{J}_\psi + (1 - \hat{p}^s_t) \gamma^t \hat{c}$
  $\hat{p}^{\mathrm{s}}_{t+1} \leftarrow \hat{p}^{\mathrm{s}}_t + (1 - \hat{p}^{\mathrm{s}}_t) \hat{\xi}_{t+1}$
  $t \leftarrow t + 1$
  ResampleParticles()
  NormalizeParticleWeights()
end
Return $\{\hat{J}_\psi, \hat{\beta}_\psi\}$

Algorithm 4.1: ExpansionApproximation

(N_{obs}) from $\mathrm{f}_{\boldsymbol{y}_t|T_{u_t}b_t,u_t}$ where u_t is chosen with respect to the local policy being simulated. The functions ResampleParticles and NormalizeParticleWeights are standard particle filtering operations that restrict the size of the parameterization by discarding particles and scale the particle weights to ensure that the sum of the weights in (2.7) is one. There are many sampling methods that can be used for ResampleParticles, but typically sampling without replacement on pmf defined by the particle weights is sufficient for many POMDP systems.

Now that the expansion operation has been defined and computation and approximation methods have been discussed, we discuss optimization of the switching law.

4.4 Optimization of the Switching Law

In this section, we develop an optimization method for a switching law of a switched policy with fixed local policies. We begin by discussing an infeasible optimization procedure, based on value iteration, that is illustrative of our general approach on this problem. In practice, we use an approach inspired by this one but based on sampling and approximation. We discuss this method, which is one of our main results, in Section 4.4.2 and a data structure to aid the method in Section 4.4.3.

4.4.1 An Illustrative but Infeasible Optimization Procedure

The goal of this section is to cast the switching law optimization in the familiar context of value iteration, and highlight the non-standard considerations that differentiate this process from standard value iteration. Recall that in Section 4.2 we developed a version of the stochastic Bellman equation suitable for computation of a policy value function for a switched policy. By analogous development, we can write a version of (4.4) for the optimal policy. Then, the value function V, restricted to a switched policy with fixed local policies, satisfies

$$V(b) = \min_{\psi \in \Psi} \left\{ J_\psi(b, \pi_\psi) + M_{b,\psi} V \right\} \tag{4.6}$$

Thus, if we can find a V that satisfies this equation we can synthesize an optimal switching law α^* using

$$\alpha^*(b) = \arg\min_{\psi \in \Psi} \left\{ J_\psi(b, \pi_\psi) + M_{b,\psi} V \right\} \tag{4.7}$$

with that V. Since we minimize over ψ, we are essentially choosing the best local policy to be switched to at every belief and using that policy in the switching law for $\overline{\pi}$.

Since we typically cannot solve for V in closed-form, one way to compute it is by value iteration (discussed in Section 3.3.1). This entails an iterative[4] approximation $\hat{V}^i$ to V where

$$\hat{V}^{i+1}(b) = \min_{\psi \in \Psi} \left\{ J_\psi(b, \pi_\psi) + M_{b,\psi} \hat{V}^i \right\} \tag{4.8}$$

Previously, we have used subscripts to notate time stage indices of the process (and conceptually related quantities). Here, we will use superscripts to denote the evolution of the value function approximation and computed policy with respect to iterations of the optimization algorithm. Once $\hat{V}^i$ has converged to satisfy (4.6), we use it in (4.7) to compute an optimal switching law.

While this algorithm is theoretically appealing, there are severe problems with both the tractability and feasibility of the exact approach. Firstly, b is a continuous space. Even taking advantage of known structure of the value function under the class of belief-feedback policies (without a restriction to using a switched policy with fixed modes), representation of V can be problematic. Secondly, computation for this procedure is prohibitively expensive. In the following section, we present a sampling-based, anytime approach to approximate this procedure.

4.4.2 Learning a Switching Law by Sampling

Th method discussed in Section 4.4.1 is theoretically sound, but not practical for implementation. The core idea of our solution approach, explicitly described in Algorithm 4.2, is to implement a version of the algorithm specified

[4]The initial condition of the recursion depends on whether the cost function is discounted or total cost with retirement.

by (4.6), (4.7), and (4.8) that can be computed in an incremental, sampled manner. The rate at which the algorithm can learn policy improvements can be increased by reusing simulations of expansions for beliefs close to one another in the belief space.

We augment our initial approach by maintaining a set of beliefs where we observe a switch may occur, denoted $\mathcal{B}^i$. Also, for each $b \in \mathcal{B}^i$, we retain the set of local policies we have expanded from that point, denoted $\mathcal{E}^i(b)$. We will also mark one policy from Ψ as special, and refer to it as the terminal policy ψ^{term}. This policy will act as the default solution for beliefs where we have not sampled a better local policy. Define the terminal cost function $\zeta(b)$ as the expected cost-to-go of using ψ^{term} at b. For the purpose of this discussion, we will consider this to be the exact expected cost-to-go, but in practice many choices for ζ are useful. Computing the terminal cost is discussed further in Appendix E. The initial condition of our iterative algorithm is then

$$\begin{aligned}\mathcal{B}^0 &= \{b_0\} \\ \mathcal{E}^0(b) &= \{\psi^{\text{term}}\}\end{aligned}$$

with the initial switching law satisfying

$$\alpha^0(b) = \psi^{\text{term}}$$

for all $b \in \mathcal{P}_b$.

At iteration i, we sample a belief b^i from $\mathcal{B}^{i-1}$ and an unexplored local policy ψ^i from $\Psi / \mathcal{E}^{i-1}(b^i)$. We then compute the expansion $\vartheta(b^i, \psi^i)$. We record the support of the expansion's stopping pdf marginalized over time as the set

$$\mathcal{B}_{b^i,\psi^i} = \left\{ b \in \mathcal{P}_b : \sum_{t=0}^{+\infty} \beta_{\psi^i}(b,t) > 0 \right\}$$

where β_{ψ^i} is conditioned on starting from b^i. This is the set of (possibly) newly explored beliefs where a switch may occur. Thus, we combine this set with $\mathcal{B}^{i-1}$. Additionally, the local policy is added our set of expanded local policies for b^i.

$$\begin{aligned}\mathcal{B}^i &= \mathcal{B}^{i-1} \cup \mathcal{B}_{b^i,\psi^i} \\ \mathcal{E}^i(b^i) &= \mathcal{E}^{i-1}(b^i) \cup \{\psi^i\}\end{aligned}$$

We then compute a $\hat{V}^i$ that satisfies

$$\hat{V}^i(b) = \min_{\psi \in \mathcal{E}^{i+1}(b)} \begin{cases} J_\psi(b, \pi_\psi) + M_{b,\psi} \hat{V}^i & \psi \neq \psi^{\text{term}} \\ \zeta(b) & \psi = \psi^{\text{term}} \end{cases} \tag{4.9}$$

for $b \in \mathcal{B}^i$. For those beliefs, this is the value function under a stronger policy restriction, i.e., restricted to the belief and local policy combinations sampled thus far. We then compute the i^{th} switching law, and thus specify a switched policy $\overline{\pi}^i$, based on $\hat{V}^i$.

$$\alpha^i(b) = \arg\min_{\psi \in \mathcal{E}^{i+1}(b)} \begin{cases} J_\psi(b, \pi_\psi) + M_{b,\psi} \hat{V}^i & \psi \neq \psi^{\text{term}} \\ \zeta(b) & \psi = \psi^{\text{term}} \end{cases} \tag{4.10}$$

Of course, the switching law is ambiguous for switching beliefs we have not touched with our optimization algorithm, so we choose to use the terminal policy, i.e.,

$$\alpha^i(b) = \psi^{\text{term}}$$

at all beliefs that have not yet been explored. As $i \to +\infty$, with an appropriate sampling algorithm we will reach all beliefs where a switch may occur with probability one.

Thus far, we have altered the ordering of how the optimal policy is computed, but so long as every belief and policy combination is sampled we have not disrupted any convergence to an optimal switching law. Here, we introduce our first approximation. Computing expansions exactly is expensive, so we employ the approximation method described in Section 4.3.2 to replace (4.9) and (4.10) with

$$\hat{V}^i(b) = \min_{\psi \in \mathcal{E}^{i+1}(b)} \begin{cases} \hat{J}_\psi(b, \pi_\psi) + \hat{M}_{b,\psi} \hat{V}^i & \psi \neq \psi^{\text{term}} \\ \zeta(b) & \psi = \psi^{\text{term}} \end{cases}$$

$$\alpha^i(b) = \arg\min_{\psi \in \mathcal{E}^{i+1}(b)} \begin{cases} \hat{J}_\psi(b, \pi_\psi) + \hat{M}_{b,\psi} \hat{V}^i & \psi \neq \psi^{\text{term}} \\ \zeta(b) & \psi = \psi^{\text{term}} \end{cases}$$

We use Algorithm 4.1 to compute $\hat{J}_\psi$ and $\hat{M}_{b,\psi}$. We have essentially replaced the exact expansion cost and switching pdf with particle filtered estimates. Thus, if an appropriate data structure is chosen to store the parameters of the

Input: b_0, Ψ, ζ
Output: $\overline{\pi}_{\text{out}}$

$\hat{\mathcal{B}}^0 \leftarrow \{b_0\}$
$\hat{\mathcal{E}}^0(b_0) \leftarrow \emptyset$
for $i = 1 :$ anytime **do**
  $\{b^i, \psi^i\} \leftarrow \text{SampleBeliefAndPolicy}(\hat{\mathcal{B}}^{i-1}, \hat{\mathcal{E}}^{i-1}, \Psi)$
  $\{\hat{\beta}_{\psi^i}, \hat{J}_{\psi^i}\} \leftarrow \text{ExpansionApproximation}(b^i, \psi^i)$
  $\hat{\mathcal{B}}_{b^i,\psi^i} = \emptyset$
  foreach $b^r \in \hat{\beta}_{\psi^i}$ **do**
    $b^s \leftarrow \arg\min_{b \in \mathcal{B}^i}[\rho(b^r, b^s)]$
    if $\rho(b^r, b^s) < \epsilon_b$ **then**
      $b^r \leftarrow b^s$
    else
      $\hat{\mathcal{B}}_{b^i,\psi^i} \leftarrow \hat{\mathcal{B}}_{b^i,\psi^i} \cup \{b^r\}$
      $\hat{\mathcal{E}}^0(b_r) \leftarrow \emptyset$
    end
  end
  $\hat{\mathcal{B}}^i = \hat{\mathcal{B}}^{i-1} \cup \hat{\mathcal{B}}_{b^i,\psi^i}$
  $\hat{\mathcal{E}}^i(b^i) = \hat{\mathcal{E}}^{i-1}(b^i) \cup \{\psi^i\}$
  $\hat{V}^i \leftarrow \text{OptimizeGraph}()$
  $\overline{\pi}^i \leftarrow \text{ExtractPolicy}(\hat{V}^i, \hat{\mathcal{B}}^i, \hat{\mathcal{E}}^i)$
end
return $\overline{\pi}_{\text{anytime}}$

Algorithm 4.2: Anytime POMDP Optimization

expansions performed, the value backup and switched policy computation can be performed with small cost at each iteration. Thus, we turn our attention to building a data structure to capture the approximate evolution of a POMDP that can be augmented incrementally for incremental learning of a switched policy.

4.4.3 Building a Switched Policy Learning Data Structure

We implement the method discussed in Section 4.4.2 with Algorithm 4.2. This is an anytime algorithm, so sampling and policy improvement continues until an external signal is sent to stop optimization, or some preset (but not algorithm specific) condition is met. A belief and local policy are sampled at each iteration. This method can be problem dependent, but care must be taken in every case to ensure that the set of beliefs that will generate an

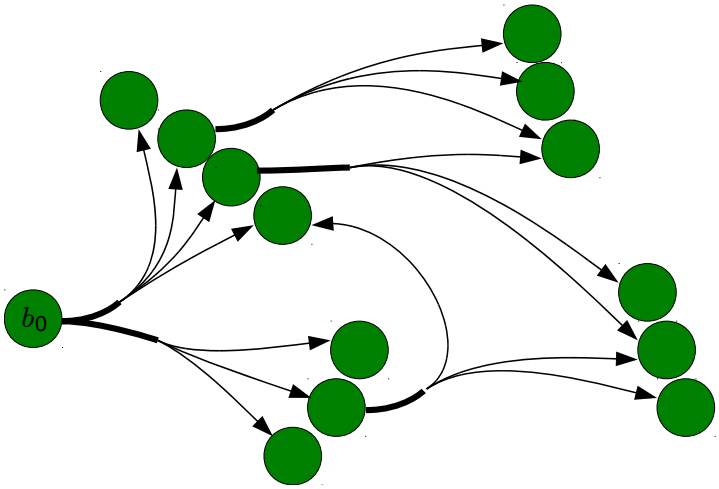


Figure 4.1: Switched Policy Learning Data Structure

effective switching law are sampled sufficiently often. We discuss sampling algorithms more in Appendix F.

Once we have sampled the domain parameters of an expansion, we use Algorithm 4.1, referred to as ExpansionApproximation, to compute the approximate switching points and expected cost until switching. Beliefs in the support of the switching points are then added to or associated with beliefs already in $\mathcal{B}$. We do this by computing the distance between each new switching point and previously reached switching points with respect to distance metric ρ. Sufficiently close beliefs, with respect to a threshold ϵ_b, are approximated to be identical. If no previously reached switching point is sufficiently close, we add this new switching belief to $\mathcal{B}$. This approximation is justified in Chapter 6. Finally, graph optimization (the OptimizeGraph operation of Algorithm 4.2) is used to compute the value function over the beliefs we have stored, which is an approximation to the true value function for the current switched policy. This approximated value function is used to choose a new switching law.

The beliefs and expansions are stored on an augmented graph structure. Consider Figure 4.1, which represents the data structure. Our augmented graph is composed of vertices (indicated in the figure with green circles), edges (indicated by thin lines), and edge bundles (indicated by thick lines). Essentially, an edge bundle has a single source vertex but many edges depart the bundle and have different target vertices. Every belief in $\mathcal{B}$ is associated to a vertex on the graph. Every expansion is associated to an edge bundle,

and each sample path in the expansion corresponds to an edge in the bundle. Each vertex, edge, and edge bundle has associated data. Vertex data includes the associated belief and computed value under the current switched policy at the associated belief. Edge bundles store the local policy used in the expansion and expected cost until switch. Edges store the number of stages elapsed in the sample path, and weight (likelihood) of the sample path. The sum of the weights on the edges emanating from a bundle is one, as this collection of sample paths represents the hyperbelief evolution of the system.

This data is sufficient to compute the value function on the graph using value iteration or Bellman-Ford (if negative cost is possible) at each iteration. In practice, we use a more efficient method that queues up a list of vertices where the computed value may be out of date. Vertices will be added to the queue if a new expansion emanating from the associated belief is added to the data structure. Then, OptimizeGraph computes the value at that vertex. If the value is improved at that vertex, then all source vertices for edges whose target vertex was just improved are added to the queue. This procedure continues until the queue is empty. Examples can be constructed where this algorithm touches as many vertices as standard value iteration, but in practice it performs significantly better in most cases and does no worse.

Simulating the system to find the expected cost until switch and switching set for expansions is, by far, the operation that requires the majority of computation resources throughout execution of the algorithm. By building this data structure, we are able to avoid re-computing expansions by trading storage space (required to keep a parameterization of all expansions on a graph) for number of computing operations (required to re-compute expansions at every optimization iteration). This data structure we build represents the evolution of the POMDP, given particular policy decisions (switching laws) are chosen.

Finally, as the number of beliefs (associated to vertices in the graph) increases, a significant amount of effort must be devoted to computing the nearest neighbor for new beliefs entering the graph. Unfortunately, as the dimension of the space is $|\mathcal{X}|$, for most large POMDPs the benefit of using nearest neighbor data structures, e.g., KD-Trees, over brute force computation is negligible (and in many cases brute force is more efficient). One can make significant progress by projecting each belief onto a parameterization,

e.g., the first n moments of the belief pdf, and using a metric on that parameterization to determine closest belief. If a small number of moments, e.g., $n < 10$, can be used without sacrificing representative quality of the beliefs, then a nearest neighbor data structure again becomes useful. As these issues appear to be a computational bottleneck for our proposed method, future exploration will be required.

Once the final switching function needs to be used, it can be extracted from the data structure. A switching function can be represented as a map from every $b \in \mathcal{B}$ to a local policy in Ψ. If a switch occurs at a belief within ϵ_b of a belief in $\mathcal{B}$, the local policy associated to the closest belief in $\mathcal{B}$ is chosen as the next local policy. If no belief is available within ϵ_b, the terminal policy is chosen as the local policy and no future switches occur.

4.5 Discussion

We have presented a method that has been verified experimentally (see Chapter 5), but important questions still need to be considered. In particular, sampling and nearest neighbor approaches seem to be direct avenues for immediate improvement in both speed of the algorithm and the corresponding performance results.

Because we use hyper-particle filtering to approximate expected cost until switch and a pdf over switching locations, we do not get the exact cost or behavior on our graph edges. Since these approximations are based on Monte Carlo sampling, in some (rare) cases, chance may promote a lower-quality edge (corresponding to the incorrect choice of a local policy at a point) above a better edge due to approximation error. Currently, there is no mechanism by which to fix this problem with additional sampling. But, the problem can be alleviated by adding steps to resample edges and refine the approximation errors.

While the approximation of condensing beliefs close to one another, i.e., considering b_s and b_r equivalent if $||b_s - b_r|| \leq \epsilon_b$, has not yet been justified, we provide an analysis of why this is (usually) a safe approximation in Chapter 6. We justify this by studying the perturbation from b to $b' = b + \Delta b$ where $||\Delta b||$ is small, and characterizing how the POMDP evolution and cost of that evolution will change over multiple stages. We have performed this analysis

for discrete systems, and a similar analysis seems promising for continuous systems. However, performing the analysis for continuous state systems still remains an open problem.

One advantage of this method is that there is an inherent possibility for re-use of computation. While a graph is built from a specific b_0, restarting the optimization procedure with the data structure already in place from a different b_0' has a potential to be significantly faster, in many cases. Thus, this has a potential to become a multi-query method, rather than optimizing for a single policy and single starting belief.

With multi-core and multi-processor computing becoming ubiquitous, an increasingly important issue in algorithms is the ability to parallelize. Our method can be parallelized by sampling multiple expansions at each iteration and evaluating those approximate expansions in parallel. In our experiments, we have seen significant performance increases from this, and it bodes well for using this algorithm on very large POMDPs requiring a supercomputer or cluster of computers to solve.

One important theoretical point is that the switching law of a switched policy could also be optimized using a standard optimization approach, i.e., optimizing single controls, on a modified version of the POMDP model. We essentially optimize a transformed POMDP process where switching choices correspond to individual controls and the range of the expansion operator specifies the one-stage transition probabilities and expected cost. However, it is important to note that this transformation process is non-trivial, as is explicitly shown in Section 4.3. This is largely due to the facts that sequences of controls specified by a local policy are not fixed a priori and local policies may evolve for differing amounts of time until switching, even when starting at the same belief over multiple trials. Thus, computing this transformation as a pre-processing to another POMDP optimization routine is typically not feasible. Our method allows interweaving computation of this transformation with the optimization process which is essential for a sampling-based approach. Essentially, we compute the pieces of the transformation we need for the optimization in a just-in-time fashion, according to the samples drawn.

Of course, one could suggest a simpler transformation, e.g., fixed control sequences for a fixed time horizon. However, we argue that both fine-grained feedback control and event-based switching conditions are often essential to a good control strategy. Thus, we place the burden of computing a compli-

cated transformation on our optimization algorithm, rather than choosing to constrain our switched policy to a family that allows re-writing of the POMDP by a simple transformation rule.

In the subsequent chapters, we present results of this algorithm on discrete systems (Chapter 5), an analysis of the robustness of the algorithm (Chapter 6), and a solution to the minimum uncertainty robot navigation problem based on this approach (Chapter 7).

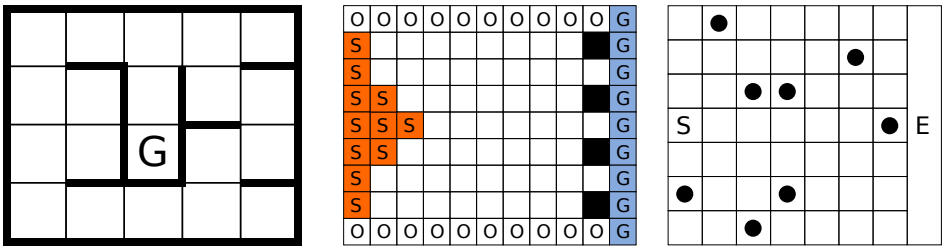
CHAPTER 5

DISCRETE POMDP EXPERIMENT RESULTS

Chapter 4 described a method for learning a switched policy for POMDP problems. This chapter presents results of this method on discrete state, observation, and control POMDP problems. These results demonstrate the viability and scalability of our method. We begin in Section 5.1 by discussing our approach as applied to several POMDP benchmark problems, which is done to demonstrate that our results are commensurate with the results of other POMDP solvers on established problems. We then present results on a more interesting system, a resource allocation problem modeled after a team of robots attempting to extinguish a forest fire, in Section 5.2.

5.1 POMDP Benchmark Problems

Often when proposing a new POMDP algorithm, investigators test their algorithms against other algorithms in the literature using benchmark problems specifically designed for POMDPs, e.g., Rock Sample, Maze20, Hallway2, Tag, Tiger-Grid, Fourth Floor, Underwater Navigation. Descriptions of these problems can be found in [28, 29, 30, 32, 31]. However, these existing benchmark problems are not well suited for evaluating our approach. The core idea of our algorithm is to exploit domain knowledge about problems and use statistical methods to analyze the result of applying local policies generated from that knowledge. Testing our method on benchmark problems requires local policies with specific knowledge about the problem at hand, while other methods will not necessarily utilize any domain knowledge. Thus, one must take care in comparing the results of our algorithm to those of other algorithms. Furthermore, the greatest strength of this method is that it allows us to consider problems that require solutions whose controls at the current stage are chosen with respect to consequences many stages in



(a) Maze20 (b) Underwater Navigation (c) Rock Sample 7×8

Figure 5.1: POMDP Benchmark Problems

the future (i.e., long horizon). The benchmark problems do not emphasize this criterion because they do not require a long planning horizon. Despite these difficulties, it is still useful to present the results of our method being applied on benchmark problems as it establishes that, given properly chosen local policies, our algorithm can achieve results commensurate with other methods in the literature. We have selected three benchmark problems and present our results in this section.

Our experimental procedure involved generating 20 policies for each problem. (Since we used a randomized sampling-based approach, each run of the optimization algorithm tends to produce a slightly different policy.) We then used Monte Carlo simulation of sample paths to generate an estimate of the expected reward using each policy. The number of trials was dynamically selected to be sufficiently large to guarantee a reasonably small confidence interval around the reported average; i.e., we continued simulation until the size of the confidence interval of the estimated expected value fell below a preset threshold. These estimates are then averaged to produce an average expected reward for policies generated using our method. This is the number we report for the problems discussed below, and they are listed in Table 5.1 alongside comparison numbers from the literature. While we have discussed cost throughout this dissertation, these problems are reward-based.

The Maze20 problem, first described in [58], models a robot navigating a maze. The robot moves on a grid and transitions between adjacent cells unless there is a wall between grid cells, and can check (probabilistically) if there are walls in the north-south directions or east-west directions. The maze is shown in Figure 5.1(a). The robot has an initial belief in being in a corner, but does not know which one, and is rewarded for moving to the goal cell which is marked 'G' in Figure 5.1(a). To solve this problem, we use

a number of local policies. Specifically, we use policies that

- act greedily, i.e., choose the control that maximizes the immediate reward.
- minimize the next-stage expected entropy.
- estimate the maximum likelihood state and, taking that estimate as ground truth, follow the optimal path to the goal.
- use the optimal control for the approximation of the problem that assumes the state will be perfectly observable after the next move (MDP approximation).
- localize the robot, i.e., specifically perform actions that allow the robot to determine its location (as much as possible).

Although these policies definitely include "domain knowledge" about the POMDP problem, we would like to emphasize that these policies can be easily chosen, defined, and implemented. None of the local policies by itself is able to compete with the switched policy, and the largest part of the optimization procedure involves composing these local policies into a switched policy.

The Underwater Navigation problem is an instance of the well-known coastal navigation problem, and is discusssed in [32]. It describes an autonomous underwater vehicle (AUV) attempting to navigate to a goal region while avoiding collision with rocks. The robot moves on a grid, and a map that has been reduced in size from the version solved (which is 51 grid cells by 52 grid cells) is shown in Figure 5.1(b). The goal region, marked with 'G', is located to the east of the (uncertain) initial position, marked with 'S.' The robot can only localize itself at the north and south of the map in the

Table 5.1: Comparison results on benchmark problems and comparison citation

Problem	Reward (Our Method)	Reward (Comparison)
Maze20	36.834	54.13* [58]
Underwater Navigation	773.32	722.59 [32]
RockSample	13.697	21.27 [32]

* Indicates a computed lower bound on expected reward under the optimal policy, not the result of a computed policy.

cells marked 'O.' The controls available to the AUV are moving north, south, east, northeast, southeast, and staying in place. From inspection of the problem, it is fairly clear the AUV should either move north, south, northeast, or southeast until it is able to localize itself, or move east until it reaches the goal or hits a rock. Thus, for this problem we choose local policies that move north, south, northeast, or southeast until the entropy of the belief drops to zero and a local policy that moves right for a fixed number of steps. Since the optimal policy is fairly simple for this problem, the algorithm converges to a good solution very quickly.

The RockSample problem models rover science exploration, and the problem is illustrated in Figure 5.1(c). A robot moves on a grid attempting to "sample" interesting rocks by moving to them and performing a sample operation. The robot earns reward if the rock it samples is "good" and is penalized is the rock is "bad." A check action can be used to ascertain if the rock is good or not probabilistically, and the measurement becomes better as the distance between the robot and the rock decreases. For this problem, we used local policies that check the rocks in the vicinity of the robot and approach rocks and sample them (if they are likely to be good).

Thus, using our method and simple local policies we are able to achieve comparable performance to the best results in the literature. On the problems discussed above, the average expected reward is at least two-thirds of the best result in the literature, and in one case exceeds the best published result. This demonstrates that our algorithm can be used to optimize some benchmark POMDPs comparably with other POMDP solvers in the literature. Of course, we do not expect to achieve better rewards than POMDP solvers that optimize stage by stage, and do not restrict POMDP trajectories along local policies. Instead, the primary advantage of our method is that it can be scaled to very large problem sizes, beyond which POMDP solvers that optimize stage by stage can successfully handle. In the next section, we demonstrate the scalability and utility of our method for solving very large POMDP systems.

5.2 The Forest Fire Fighter's Problem

In this section, we present a POMDP problem we have formulated and then demonstrate our algorithm's effectiveness. The target application for this method is planning policies for a group of robots attempting to control a random process. We tested our algorithm on a multi-robot task on a grid workspace. The robots' goal is to coordinate to manipulate a stochastic process inspired by the spread of a fire evolving on the robots' workspace. The robots will attempt to contain and extinguish the fire. The size (state, control, and observation space dimension) of this POMDP is much larger than other discrete POMDP models presented in the literature and, to the authors' knowledge, no result has been published on a system this large to date. We first describe the formulation of the problem at a high level and then present the results of application of our algorithm to this POMDP.

This problem provides an excellent example of how a seemingly innocuous planning problem can become combinatorially complex. Consider the example of 10 robots on a workspace of a 25×25 two-dimensional grid, trying to optimize a cost function, a problem with small size in many robotics contexts. If every robot can move to any of its neighboring cells, the set of controls has 2^{30} possibilities to evaluate. If we have no additional way to prune this set, a naive or brute-force approach will require checking 2^{30} controls to compute a 1-step optimal policy. In fact, if we sample this set uniformly without replacement we will need 2^{28} samples in order to achieve only a 0.25 probability of finding the optimal one-step action. While the problem is already expensive for one stage, if planning is to occur over multiple stages we must consider an exponential branching of possibilities due to the exponentially growing set of possible sequences of observations. Despite these difficulties, our method is able to make significant progress on this hard POMDP problem.

5.2.1 System Model

We model the workspace as an $M_1 \times M_2$ grid and place N robots on the grid. Thus, a state $x \in \mathcal{X}$ specifies whether each grid cell is on fire and, for each robot, the grid cell in which it resides. This implies the number of states is $2^{(M_1 \cdot M_2)} \cdot N(M_1 \cdot M_2)$. For example, for the case of 10 robots on

a 25×25 grid there are approximately 8.7×10^{191} states. We model the robots acting deterministically, but each robot has a limited range of view. Thus, even when sensor readings are combined, the state of the fire typically remains only partially observed. To discuss the problem in a concise manner, we introduce a number of operators to extract specific information from a state. The indicator function $c^{ij}(x)$ specifies if grid cell at (i, j) is on fire for state x. The operator $r^n(x_t)$ returns the location of robot n in the grid. We define $R^{ij}(x)$ to be the indicator function that returns one if and only if $r^n(x) = (i, j)$ for some $1 \leq n \leq N$.

Each robot is allowed to move according to the equation

$$r^n(x_{k+1}) = r^n(x_t) + u_t^n$$

where u_t^n can take values that allow movement to the eight-point connected cells on the grid. Note that there is no noise in the robot motion model, and the uncertainty in the process model comes from the stochastic evolution of the fire.

We model the fire as a simple graph diffusion process. If a cell catches fire, it will remain on fire until it is extinguished by a robot. This can be expressed in terms of the state evolution as

$$\mathrm{p}\left(c^{ij}(\boldsymbol{x}_{t+1}) = 1 | R^{ij}(\boldsymbol{x}_t) = 0, c^{ij}(\boldsymbol{x}_t) = 1\right) = 1$$

If a robot does reach a cell that is on fire, it extinguishes the fire on at that cell with probability one. This can be expressed as

$$\mathrm{p}\left(c^{ij}(\boldsymbol{x}_{t+1}) = 1 | R^{ij}(\boldsymbol{x}_t) = 1\right) = 0$$

Finally, if a cell is not on fire and no robot is present, it may catch fire with probability related to fire in the four-point connected adjacent grid cells. Specifically,

$$\mathrm{p}\left(c^{ij}(\boldsymbol{x}_{t+1}) = 1 | R^{ij}(\boldsymbol{x}_t) = 0, c^{ij}(\boldsymbol{x}_t,) = 0, \boldsymbol{x}_t = x\right) = \sum_{(a,b)\in\mathcal{G}} \left[s(a, b, i, j)c^{ab}(x)\right]$$

where $\mathcal{G}$ is the four-point connected neighborhood around cell (i, j). The function s maps two grid locations to $[0, \frac{1}{4}]$ and encodes the rate of spread

between neighboring cells. Using these probabilistic relationships for all cells and robots allows us to formulate the process model for the POMDP.

Each robot can see, deterministically, a limited distance o_c around itself and each visible cell can be observed to be on fire or not. Thus, an observation is an $M_1 \times M_2$ length binary vector. Let $y^{ij}(x_t)$ indicate the binary value associated to cell ij; then

$$y^{ij}(x) = \begin{cases} c^{ij}(x) & \text{if } ||(i,j)-(i',j')||_\infty < o_c \text{ for any } (i',j') \text{ s.t. } R^{i'j'}(x) = 1 \\ 1 & \text{otherwise w.p. } \frac{1}{2} \\ 0 & \text{otherwise w.p. } \frac{1}{2} \end{cases}$$

Thus, robots can observe the fire perfectly if it is close, but have no information about cells that are far away from all robots. The observation space is large, i.e., on the order of $2^{(o_c^2-1)N}$. Using these probabilities we can generate the observation model for the POMDP.

5.2.2 Cost

Qualitatively, we want the robots to exhibit behavior to achieve two complementary tasks. The robots should, if possible, completely extinguish the fire. While doing so, the robots should protect more valuable regions of the grid while completing the first task, or focus solely on this goal in the case where the first task cannot be completed. We express this in terms of a one-stage state cost function

$$l^{\mathcal{X}}(x) = \sum_{(i,j)} a^{ij} c^{ij}(x)$$

where $a^{ij} \in \mathbb{R}^+$ is a set of values that weights the relative importance of different grid cells. If we store the binary components of the state corresponding to fire at grid cells in a vector, this can be expressed as a vector multiplication. This can be converted to our standard one-stage belief cost by taking the expected state cost.

$$l(b,u) = \sum_{x \in \mathcal{X}} l^{\mathcal{X}}(x) b(x)$$

We can then write this as a total cost with retirement problem, i.e., (3.4), choosing the retirement cost to correspond to the amount of fire remaining.

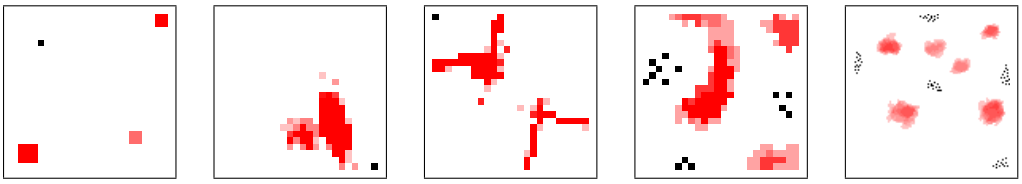
5.2.3 Belief Representation

The full belief vector is large (length the size of the state space) and the number of nonzero components, the support of the pmf, typically grows exponentially as time evolves. Even though this could be represented as a vector, for implementation this approach is not feasible. Instead, we utilize a particle filtering approximation (see Section 2.4) to both contain the growth of the support of our belief vector and also maintain a representative set of samples. The only modification to the standard particle filtering algorithm is to take multiple samples from the process model in the transition probability phase and to sample without replacement.

5.2.4 Policies

The policies we have chosen are simple to compute and designed to be nearly decentralized; i.e., each robot acts with minimal need to check the controls of other robots. However, policy sampling operations, such as choosing targets for robots, are designed to create high-level cooperation between robots when a new policy is applied, e.g., coordinating which robots will attack different portions of the fire.

For brevity, we will not give explicit algorithmic specification of our policies, but instead describe them at a high level. We include parameterized policies that direct robots to target regions in the space. Those targets are sampled in a number of different ways, one example being sampling on the perimeter of the fire. Other policies direct robots to the nearest grid cell where there is a nonzero probability of fire, instruct robots to stay in place unless they are able to attack fire in the immediate vicinity, move robots around the perimeter of the fire, and attempt to cluster the robots at various targets in the workspace. The terminal policy uses a brushfire expansion, [56], on the workspace grid. A potential function is computed that draws robots to cells with nonzero probability of fire. Robots follow this gradient to find and extinguish fire. Note that the terminal policy we used is not



(a) b_0 for Experiment 1 (b) b_0 for Experiment 2 (c) b_0 for Experiment 3 (d) b_0 for Experiment 4 (e) b_0 for Experiment 5

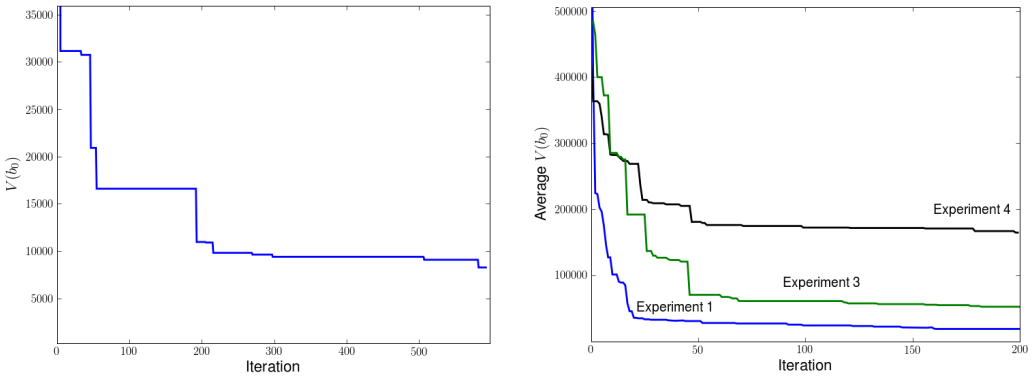
Figure 5.2: Initial Belief States for Experiments

the typical greedy policy with respect to the cost function. This is because when no robot is directly adjacent to a cell containing fire, the gradient in the cost function is zero. This prevents the robots from making progress at a large number of belief states and severely limits the usefulness of the policy. However, our terminal policy is equivalent to the greedy policy at all places where the greedy policy has a nonzero gradient in cost. The retirement condition for the system occurs when we are applying the terminal policy and (i) the probability of fire in all cells is zero for the current belief or (ii) the sum of the expected value of fire over all cells exceeds some proportion of the number of workspace cells, i.e., the robots will not be able to recover and fire has taken over the workspace.

5.2.5 Experiment Results

To demonstrate the method, we chose four initial conditions where $M_1 = M_2 = 25$, $N = 14$ (Experiments 1-4) and one where $M_1 = M_2 = 100$, $N = 70$ (Experiment 5). The initial conditions are shown in Figure 5.2. In these figures, the intensity of the red values in the table represent the probability of fire with red corresponding to probability one. A lighter red corresponds to lower probability values, but for visualization purposes the probability is scaled between intensity of one half and one instead of zero and one. A black cell indicates the cell contains at least one robot. A white cell indicates zero probability of fire and no robot is present. The transition probabilities for fire from cell to cell were uniform throughout the workspace, i.e., $s(i, j, a, b)$ constant for all (i, j), (a, b).

We ran a number of trials on each experiment setup and the results of Experiments 1-4 are reported in Table 5.2. Unfortunately, finding a lower bound on cost by computing the value for the fully observed, MDP version



(a) Result of single long run

(b) Averaged result over all trials of Experiments 1, 3 and 4

Figure 5.3: Plots of $V(b_0)$ vs. Iteration

Table 5.2: Cumulative results of Experiments 1-4

Experiment	1	2	3	4
Number Trials	10	6	10	10
Number Iterations	200	50	200	200
Avg. Greedy Cost	782,870.8	56,151.1	427,015.0	484,236.9
Avg. Final Cost	18,860.5	34,066.5	164,694.0	52,459.5
Std. Dev. Final Cost	8,370.4	10,918.7	8,500.1	13,071.4
Avg. Time/Iteration[1]	565.2 (s)	514.8 (s)	213.1 (s)	391.9 (s)

of the problem is still prohibitively expensive. Thus, we use the expected cost with a greedy policy as a baseline for comparison. Plots of the average expected cost versus number of iterations, averaged over all trials, for Experiments 1, 3, and 4 are shown in Figure 5.3 (b). In Figure 5.3 (a) we show the expected cost versus the number of iterations for a single trial with an extended number of iterations. The tests were conducted on workstation class computers.

The experiments demonstrate a significant reduction in expected cost from the greedy policy in each of the experiments. Moreover, the starting conditions for each of the experiments vary greatly and the observed convergence of the value suggests the method performs as desired. The resulting policies often represent behavior that may be counterintuitive. For instance, in Experiment 2 the robots start beneath the fire in the right-hand corner. The initial greedy policy attacks the fire by moving directly towards the fire, from

[1]Only trials computed on identical hardware used.

the bottom. However, a plan is found that manages to extinguish the fire with a significantly lower cost. The robots are able to achieve this reduction in cost by first circling above the fire on the top and right, and then "pushing" the fire down and into the bottom. This type of behavior requires controls be determined by looking forward a significant number of stages into the future.

Even though the examples in the first four experiments represent systems that require significant foresight in the planning algorithm, we also ran an experiment on an example with a 100×100 grid. In this problem, we not only increased the size of the workspace, but also significantly increased the number of robots, size of observation space, and initial amount of fire present in the workspace. For a single trial, over 30 iterations, we noticed a decrease from the greedy policy value of 2.60×10^7 to 1.70×10^7. The average time per iteration was 1774.5 (s), but note that this figure is averaged over a small number of iterations and should be only taken as a relative indication of the computational requirements of scaling the problem up.

CHAPTER 6

SENSITIVITY OF DISCRETE STATE POMDP EXPANSIONS

Simulating an expansion is computationally expensive. Thus, it makes good engineering sense to re-use the data provided by a simulation to the maximum extent, and in Chapter 4 we proposed a method to learn a switched policy that was based on approximation of POMDP evolution, sampling, and re-use of simulations of POMDP evolution. In order for such a method to be useful in practice, it must be the case that small perturbations to the beliefs we analyze will not drastically change the result of the expansion operator.

This is important, firstly, because when we approximate beliefs we unintentionally introduce small perturbations to the true location an expansion should start from. Secondly, if expansions are very sensitive, it may take a very long time to get the "right" sample to an inexpensive policy. Finally, we intentionally introduce perturbations when we collapse nearby beliefs onto one another to re-use simulation during the learning process. Thus, for our approach to be successful, it must be the case that small perturbations in the belief space will not disturb the overall optimization method. We have found this to be the case experimentally, and can demonstrate quantitatively why this phenomenon occurs.

In this chapter, we develop sufficient conditions to verify when re-use of an expansion for nearby starting beliefs is appropriate, and specifically demonstrate how the cost and stopping hyperbelief estimates vary as a result of the perturbation. We demonstrate for discrete state systems that the expansion operator ϑ is typically insensitive to perturbations in the position of the initial belief. Specifically, we assert that if b and b' are near to one another in the belief space, applying local policy ψ to each will yield a similar result, i.e., J_ψ and β_ψ will be nearly identical. We derive a set of sensitivity equations that characterize the effect of small perturbations in the belief space, and discuss how this can be used to ensure the robustness of Algorithm 4.2. For the purpose of exposition, we will commit to the L_1 metric on $\mathcal{P}_b$ for

analysis, but in general any metric can be chosen.

A large part of our gains in Algorithm 4.2 are made by performing temporal abstraction over expansions. Once an expansion is parameterized and inserted into the graph, computing the value (under the current best switched policy) at beliefs in the graph can be done in $O(1)$ time with respect to the number of stages encapsulated by the expansion. In order for the sensitivity equations to be useful within this framework, they must also be succinctly parameterizable and computation of the sensitivity with respect to the perturbation must also occur in $O(1)$ time. These constraints are both met by the development in this chapter.

We use a constructive approach to develop the sensitivity equations. We begin by looking at the sensitivity of the terminal belief of, and cost-to-go along, a single sample path. Then, we extend the equations for a sample path to perturbations in the hyperbelief space.

Although these results can be used, in theory, as an algorithmic component of the learning method, we have found the computational expense tends to outweigh any performance gains of the method. This is because, typically, expansions are very insensitive and small approximations tend to just work without explicit safety checks. Furthermore, since perturbations are small, the sensitivity equations do not increase performance, except in the case where a catastrophic mistake is made that propagates all the way to the policy generated by the algorithm. This case tends to be rare in practice. Thus, although we have tested (and briefly discuss) how these equations can be used to improve the algorithm, we offer this analysis mainly as an explanation of why the algorithm tends to work in practice without sensitivity bounds.

6.1 Sensitivity of Sample Paths

Define a belief perturbation vector Δb_0 to be any vector in $\mathbb{R}^{|\mathcal{X}|}$ where $b_0 + \Delta b_0 \in \mathcal{P}_b$. The space of perturbation vectors will vary based on the position of the initial belief being perturbed. We introduce the notation

$$\bar{\phi}_{y,u} := O_{y,u} T_u$$

to refer to the unnormalized belief transition matrix.[1] Substituting $b_0 + \Delta b_0$ into (2.4) and expressing the left hand side as the original result plus perturbation, we get

$$b_1 + \Delta b_1 = \frac{\bar{\phi}_{y_1,u_1}(b_0 + \Delta b_0)}{\mathbf{1}'\bar{\phi}_{y_1,u_1}(b_0 + \Delta b_0)}$$

By rearranging terms and concatenating the one stage perturbation expressions (see Appendix C), we can express the resulting perturbation along a specific t-stage sample path in terms of initial belief and perturbation as

$$\Delta b_t(\Delta b_0) = \frac{\bar{\phi}_{0:t}\left(I - \frac{\bar{\phi}_{0:t} b_0 \mathbf{1}'\bar{\phi}_{0:t}}{\mathbf{1}'\bar{\phi}_{0:t} b_0}\right)\Delta b_0}{\mathbf{1}'\bar{\phi}_{0:t}(b_0 + \Delta b_0)} \tag{6.1}$$

where

$$\bar{\phi}_{0:t} := \bar{\phi}_{y_t,u_{t-1}}\bar{\phi}_{y_{t-1},u_{t-2}} \cdots \bar{\phi}_{y_1,u_0}$$

For the moment, we will assume that the perturbation is sufficiently small so that the sequence of controls (generated by a belief-feedback policy) does not change. Equation (6.1) explicitly depends on this assumption, because $\bar{\phi}_{0:t}$ will change if one of the controls in the sequence changes. We will address this assumption later in this section. The expression $\Delta b_t(\Delta b_0)$, for a specific b_0, maps Δb_0 to the subset of vectors in $\mathbb{R}^{|\mathcal{X}|}$ whose components sum to zero and have no element with magnitude greater than one. We will omit the argument to Δb_t when it is clear from context.

The denominator of (6.1) is a scalar normalization factor. Thus, it may be omitted initially and recovered later by projecting the resulting unnormalized belief back onto the belief space, which is a simplex. This means we can construct $\bar{\phi}_{0:t}$ iteratively as we simulate a sample path using

$$\bar{\phi}_{0:t} = O_{y_t,u_t} T_{u_t} \bar{\phi}_{0:t-1}$$

This requires only two (typically sparse) matrix multiplications per stage. The quantity $\mathbf{1}'\bar{\phi}_{0:t}(b_0 + \Delta b_0)$ represents the likelihood of the sample path, i.e., the likelihood of the sequence of observations observed along this sample path, and $\mathbf{1}'\bar{\phi}_{0:t}\Delta b_0$ is the change in likelihood of a particular sample path

[1] Specifically, comparing to (2.4), there is no normalization term to account for the probability of the observation y.

sequence.

Computing the sensitivity of the cost-to-go along a sample path is not as straightforward as the process evolution. For the simplicity of discussion, we will discuss the case where cost is linear in b, i.e.,

$$l(b, u) = l(u)'b$$

where $l(u)$ is column vector of dimension $|\mathcal{X}|$, and $'$ denotes the transpose operator. In general, the overall approach will be useful for any l continuous in b. Inserting $b_0 + \Delta b_0$ into (3.2), we find

$$j_{\bar{t}}(b_0, \Delta b_0, \pi, y_{1:t}) = \sum_{t=0}^{\bar{t}} \gamma^t l(u)'(b_t + \Delta b_t) = j_{\bar{t}}(b_0) + \sum_{t=0}^{\bar{t}} \gamma^t \; l(u)'\Delta b_t \qquad (6.2)$$

where Δb_t is given by (6.1) and $u = \pi(b_t + \Delta b_t)$. This expression gives an exact compution of perturbed cost, but typically (6.2) cannot be used directly in practice. To use (6.2) we must compute a term of the summation for every stage of the sample path. Although possible, this is clearly not compatible with the temporal abstraction requirements of our learning algorithm. Furthermore, storing a belief perturbation matrix for every stage is typically not practical for many sample paths encompassing a sizable number of time steps.

To avoid this difficulty, we use a Taylor series approximation of the term of (6.2) that results from the effect of the perturbation. One can refine the approximation with arbitrarily many terms, but here we use a second order Taylor expansion.[2] Define $\Delta j_{\bar{t}}$ and g as

$$\Delta j_{\bar{t}}(b_0, \Delta b_0) := \sum_{t=0}^{\bar{t}} \gamma^t \; g(b_0, \Delta b_t) := \sum_{t=0}^{\bar{t}} \gamma^t l(u)'\Delta b_t$$

for exposition purposes. Then, by computing Taylor expansions of g at each time step and grouping terms appropriately, we arrive at a Taylor expansion

[2]The primary motivation for using at least a second order approximation is that the Hessian matrix is required for our sufficient conditions for insensitivity. Additional precision is typically not required, particularly as computing and storing higher order Taylor terms may become prohibitively expensive for systems with large state spaces.

for the perturbed cost difference

$$\begin{aligned}\hat{j}_{\bar{t}}(b_0, \Delta b_0) &= j_{\bar{t}}(b_0) + \Delta j_{\bar{t}}(b_0, \Delta b_0) \\ &= j_{\bar{t}}(b_0) + d(b_0)'\Delta b_0 + \frac{1}{2}\Delta b_0' D(b_0)\Delta b_0 + O(||\Delta b_0||^3)\end{aligned}$$

where $d(b_0)$ and $D(b_0)$ are a vector and matrix, respectively, whose construction depends only on b_0 (and the POMDP model). Thus, to compute approximate cost perturbations (with error on the order of at most $(||\Delta b_0||^3)$, we are only required to store a matrix and vector per sample path. Furthermore, the matrix and vector can be computed iteratively while simulating a sample path. The derivation of these terms is given in Appendix D.

These sensitivity equations are complicated by the use of closed-loop control policies, e.g., in expansions. We have developed the sensitivity equations assuming the control will not change due to perturbation, which gives us an an implicit assumption that $\pi(b_t + \Delta b_t) = \pi(b_t)$ for small Δb_0. This assumption is present in (6.2), as a different cost vector will be present for different control, and in (6.1), as a different control yields a different belief transition matrix and, in general, may invalidate the sensitivity equation for the remainder of the sample path. We will address this assumption in Section 6.3.

6.2 Sensitivity of Expansions

For POMDPs with discrete and finite observation space, the hyperbelief will have only a finite number of support points for finite time horizon T. Thus, we can use (6.1) directly to characterize the perturbation of those support points and the likelihoods at those points. For POMDPs with continuous observation space, we can use (6.1) to express the perturbation to the continuous hyperbelief pdf point-wise. It is important to note that the equations below hold exactly for discrete and finite observation space POMDPs, but typically they will be used as an approximation, as one will typical employ hyper-particle filtering to approximate the evolution of the system. Thus, even for continuous observation space POMDPs these equations are a viable way to compute hyperbelief perturbations in approximation.

We can characterize the sensitivity of the hyperbelief process using (6.1). This requires us to recognize that for a discrete POMDP, the hyperbelief can

be equivalently characterized as a set of beliefs resulting from a collection of sample paths and the likelihoods of those sample paths, i.e.,

$$\beta_t(b_0, \Delta b_0) = \sum_i [w_t^i + \mathbf{1}' \bar{\phi}_{0:t} \Delta b_0] \delta \left(b_t^i + \Delta b_t^i(\Delta b_0) \right)$$

Sensivity for the expected cost function can also be written as an expectation over a set of sample paths.

$$\begin{aligned} J_T(b_0, \pi) &= \mathrm{E}_{\boldsymbol{y}_{1:T}|b_0,\pi} \left[\sum_{t=0}^{T} \gamma^t l \left(\pi(\boldsymbol{b}_t) \right)' \boldsymbol{b}_t, |b_0, \pi \right] \\ &= \sum_{y_{1:t}} \left[\mathrm{p}_{\boldsymbol{y}_{1:t}|b_0,\pi} \left(y_{1:t} | b_0, \pi \right) \cdot \sum_{t=0}^{T} \gamma^t l \left(\pi(b_t) \right)' b_t \right] \end{aligned} \tag{6.3}$$

Again, we encounter the same problem of summing over multiple stages. We can again circumvent this problem by using a weighted combination of Taylor series approximations of sample paths, or by performing a Taylor series approximation on (6.3) directly. The equations are omitted here for brevity, but they can be determined using a procedure identical to the sample path case.

Particular attention must be paid to normalization operations in approximation algorithms, such as hyper-particle filtering. The normalization serves to ensure the sum of the weights, at every time step, is one. The sensitivity equation approximation should also preserve this relationship as closely as possible. Otherwise, perturbations may scale the approximation of cost to be either larger or smaller than appropriate, and the approximation will fail.

6.3 Sufficient Conditions for Insensitivity

Thus far in this section, we have developed a set of tools that allow us to analyze the effect of a perturbation to b_0 on an expansion. However, we have not discussed how to determine if approximating $b' = b + \Delta b$ with b is appropriate. Experimentally, we have observed that sample paths sharing the same set of observations (and thus hyperbeliefs starting from different single belief states) often tend to get closer to one another as they evolve. However, there are a few conditions that cause sample paths beginning from

b and b' (that are close to one another) to diverge.

The first case involves the fact that there are some points in the belief space that are sensitive to initial position. These tend to be characterized by certain boundaries in the belief space. When these boundaries are crossed, a qualitatively different POMDP evolution emerges. Since (6.1) is an exact characterization of perturbation, we can ensure that sample path behavior will not change qualitatively by examining how the perturbation evolves. We will develop an under-approximation to this region where this is the case by ensuring

1. perturbations contract along all sample paths, i.e., $||\Delta b_t|| \geq ||\Delta b_{t+1}||$.

2. likelihoods of sample paths change by a bounded amount along all sample paths, i.e., $||\mathbf{1}'\phi_{0:t}b_0 - \mathbf{1}'\phi_{0:t}(b_0 + \Delta b_0)|| \leq \epsilon_w$.

Algorithmically, we can check these conditions for each sample path without checking individual points if the region in which we use the sensitivity equations is small. Define neighborhood $\mathcal{B}_\epsilon$ around b_0, in which the sensitivity data structure may be used. We will check all points inside $\mathcal{B}_\epsilon$ to ensure the two conditions above are met. The procedure to do so at each filtering stage (in the expansion) is:

1. Let $p = \min_{b \in \mathcal{B}_\epsilon} [\mathbf{1}'\phi_{0:t}b]$. If $\lambda_{\max}(\phi_{0:t})/p < 1$ (where $\lambda_{\max}(A)$ refers to the maximum eigenvalue of the matrix A), contraction is guaranteed.

2. Let $\Delta w^* = \max_{b \in \mathcal{B}_\epsilon} ||\mathbf{1}'\phi_{0:t}b_0 - \mathbf{1}'\phi_{0:t}b||$. If $\Delta w^* < \epsilon_w$, condition 2 is guaranteed.

The minimization and maximization are over linear functions, so a properly chosen[3] $\mathcal{B}_\epsilon$ will ensure optima can be found. These conditions can be checked as the POMDP is simulated. We can also guarantee the quality of our cost approximation since the quantity Δb_t can be computed exactly, and one can establish upper and lower bounds on cost.

The second case that causes the evolutions of b and b' to diverge is due to boundaries in the policy function. These discontinuities arise from switches in the control returned, i.e., $\pi(b) = u^a$ but $\pi(b') = u^b$ only if moving from b to b' crosses a boundary in π. This case is significantly harder to analyze

[3]For example, using the L1 metric $\mathcal{B}_\epsilon$ is a convex region so one must only check a finite number of corners to find the max.

due to the often complex geometry in policy functions and the number of opportunities the sample path has to cross a boundary. If the perturbation at any stage pushes the sample path process across a boundary in the policy function, the control can radically change the sample path evolution and the belief predicted by the sensitivity equations will be erroneous.

This might seem to suggest that this analysis is only valid for policies that are trivial or open loop, i.e., policies that map to the same control for every belief or do so for a fixed belief at each time step. However, in practice, the sensitivity approach seems to work well for closed loop policies as well. We believe this is due to a combination of factors. First, the majority of sample path evolutions do not come close enough to a switching surface to be pushed across by a small perturbation. Secondly, the goal of closed-loop policies is typically regulation. Thus, for many of the policies one encounters in practice, crossing a switching surface will help to attenuate disturbances or perturbations in the belief process. These are not necessarily justifications that the control law can be ignored, and mathematical counterexamples can, of course, be generated that can create arbitrarily large errors. But, since the effects are typically very small in the systems on which we have experimented, these points are noteworthy from an engineering standpoint.

That being said, there is more that can be done to guarantee the validity of the sensitivity equations. We have already demonstrated how to select a region $\mathcal{B}_\epsilon$ that guarantees contraction of two sample paths (assuming identical controls were applied). We can reduce the size of $\mathcal{B}_\epsilon$ to ensure perturbations cannot cross policy switching surfaces. For some policies, the minimum distance from the process to a switching surface can be computed at every time step. This can be propagated backwards along the sample path steps to select $\mathcal{B}_\epsilon$ upon reaching stage zero. Of course, this operation is highly dependent on the geometry of the policy, and will typically need to be tediously implemented for each individual policy. Also, the representation cost of such a region is likely to be high in most cases. However, once these policy-specific surfaces of discontinuity have been manually defined, the conditions can again be checked automatically.

6.4 Improving Learning with Sensitivity

In Algorithm 4.2 we merge nearby beliefs without explicitly checking if this approximation is valid. We can use the analysis of this section to validate the robustness of this procedure, as well as to improve our estimate of expected cost. We emphasize that since the perturbations in cost will necessarily be small in insensitive regions, the important improvement is with regard to algorithm robustness.

One can use the sufficient conditions of Section 6.3 to adaptively adjust the region defined by ϵ_b in Algorithm 4.2. Thus, we will only re-use expansions in insensitive portions of the belief space. When performing the OptimizeGraph procedure, we must also propagate the perturbation forward along optimal trajectories to verify that connecting a path through the graph does not push the process outside the region of validity for any expansion along that path. If this does occur, additional simulation will need to be performed, or the corresponding switching strategy will need to be discarded.

6.5 Discussion

Experimentally, we have found that, for many problems, the amount of (computational and code complexity) overhead introduced by the procedure of this section does not justify its inclusion because of the extremely low frequency of approximation mistakes of the naive algorithm that cause poor results. However, for other systems this analysis can be very important for verifying the robustness of the approximation. Aside from the typically inherent insensitivity of expansions, we postulate that the values of ϵ_b we have used are sufficiently small to negate much of the usefulness of the sensitivity procedure outlined in this chapter. For this reason, the core purpose of this chapter is to explain why our approach is sound. However, since increasing ϵ_b adaptively can potentially provide large efficiency and performance increases, this is an important area of future investigation.

CHAPTER 7

MINIMUM UNCERTAINTY ROBOT NAVIGATION

In this chapter, we discuss a particular application of our switched policy POMDP optimization algorithm: minimum uncertainty robot navigation. This problem, concisely, entails computing a feedback policy for robots with stochastic motion and observation models that minimizes a cost criterion that penalizes probability of collision and the uncertainty around a goal configuration. We present two new methods for finding minimum uncertainty plans.

While this problem has a long history and has been considered by many other researchers (see Sections 7.1.1 and 7.1.2), the fundamental barriers to stochastic robot navigation are the same as the POMDP optimization problem. The ideal model for a robot with noisy sensors and a stochastic motion model is a POMDP, but optimizing a feedback policy with respect to such a model is typically hard due to dimensionality and pdf representation. For typical robot models and environments, the belief pdf will become difficult to represent because of nonlinear robot models, realistic sensor models, and obstacle constraints. Adding this to the typical motion planning problem complexity (see Section 7.1.1), computing policies using a POMDP model can become extremely difficult.

To circumvent these difficulties, most minimum uncertainty navigation methods make one or more of the following simplifying approximations:

- They plan in an augmented configuration space, and use state feedback (in conjunction with state estimation) rather than belief feedback.
- They use a concise (and often insufficient) belief parameterization that is only representative of the true belief under a very limiting set of conditions.
- They assume full knowledge of future observations, or consider only the maximum likelihood observation during optimization to circumvent

considering the full hyperbelief evolution of the system.

- They coarsely discretize the problem model.

These methods often yield useful results, but can also make poor optimization choices. For example, using only state estimation or parameterizing the belief with a unimodal pdf is particularly dangerous when the true belief is multi-modal. Considering only the maximum likelihood observation will often lead the robot to situations at run time that have not been analyzed by the planning algorithm. Finally, coarsely discretizing the problem model, without careful consideration of the problem structure in the belief and hyperbelief transition laws, can ignore important aspects of the model that can lead to greatly improving solution quality. These methods do work, but under specific circumstances or with limited effectiveness.

We have developed a POMDP optimization method (outlined in Chapter 4) that is capable of optimizing a POMDP that fully captures the stochastic robot navigation problem. Thus, we present a pair of approaches, based on this method, to the minimum uncertainty robot navigation problem. Before presenting our work, we briefly survey robot navigation in Section 7.1.1 and discuss why minimum uncertainty robot navigation is hard. We present specific stochastic robot navigation techniques in Section 7.1.2. With the requisite background, we then discuss our general approach to attack the problem in Section 7.2. We then present our two specific methods to solve this problem. The first, discussed in Section 7.3, is based on using a roadmap heuristic and is mainly presented as a comparison to other techniques in the literature. The second, presented in Section 7.4, incorporates information theory heuristics. We believe this second method is an important step in building a general purpose minimum uncertainty navigation planner for general robot systems.

7.1 Related Research

In this section, we give a brief overview of the robot navigation problem. We first discuss path and motion planning problems, which are for deterministic robots. We then discuss approaches to this problem for stochastic models.

7.1.1 Robot Navigation

The *navigation task* is one of the central focuses of research in the robotics community. Navigation is the problem of finding a collision-free motion for a robot to transition from one state to another [56]. Algorithms that solve this problem are useful to automate the movement of robots, parts, or other geometric objects through arbitrary environments. The most basic navigation problem is the *path planning problem*. This problem is concerned with moving a robot from one configuration to another with only obstacle constraints, i.e., no constraints on the movement of the robot in free space. This problem was shown in [59] to be PSPACE-complete. This paper also gives an algorithm that, in the general case, is the best known complete solution to the general path planning problem. It has running time

$$p^m \cdot \log p \cdot w^{O(m^2)}$$

where p is the number of polynomials required to represent the robot's free configuration space, w is the highest degree of any of those polynomials and m is the number of degrees of freedom of the robot. In addition to the dependence on the number of degrees of freedom of the robot, there is also a nonlinear dependence on the size of the environment. Thus, even without realistic motion constraints or uncertainty, complete solutions to this problem are expensive.

When differential constraints are introduced into the model of the robot's motion, the task of finding a collision-free trajectory from one state to another is known as the *motion planning problem*. Due to the underlying complexity of the path planning problem combined with the addition of differential constraints, sampling-based methods, e.g., [60, 61], and other approximation or heuristic methods are commonly used. Because these algorithms are probabilistic, they cannot be shown to be complete in the conventional sense. Thus, they rely on probabilistic completeness; i.e., if a solution exists, the probability the algorithm fails to find it goes to zero as the number of samples goes to infinity. These tools are probabilistic, but find paths for deterministic systems.

A natural progression from these algorithms is to attempt to find an optimal trajectory, rather than any feasible trajectory. Thus, for deterministic

robot systems, motion planning research has progressed from considering only feasibility to finding optimum paths where the quality of paths is measured by some metric of minimum effort. Analogously, in the case of a stochastic, partially observable state robot model, different control policies will result in different likelihoods of collision with obstacles and uncertainty at the goal position. A natural goal is to increase the reliability of robot systems by synthesizing control policies that seek to minimize these criteria.

Several difficulties come along with this problem. Since the state is partially observable, we will only be able to control a belief of the robot's state. Thus, planning must be done in the belief space treating the model as an MDP. As discussed throughout this thesis, the lifting process from state to belief space (and belief to hyperbelief) adds significant dimensionality to the planning and optimization problems.

Most realistic sensor models are not invariant to the robot's state. For example, range sensors are typically more accurate when close to obstacles, beacons become noisier as the distance between the robot and beacon increases, and some regions of the workspace will allow cameras to capture images that are rich with features while others are largely devoid of identifiable points. As a result of this dependence on state, the expected effectiveness of a measurement reducing uncertainty will depend on the current belief, and thus the trajectory of the system.

Differing quality of sensor information is not the only factor affecting the quality of a trajectory. Different trajectories will also induce different levels of uncertainty due to process noise. Even if we could evaluate all paths through the configuration space and determine the one that provides the most localization information, we cannot hope to follow that path exactly because of uncertainty in the robot's process model. Thus, in general, a minimum uncertainty path optimization involves a trade-off between exploring regions of the state space where more information about the state can be gained versus taking a path to the goal that introduces the smallest amount of process noise into the estimate of the system's state.

The uncertain system process model also implies that a straightforward extension of sampling-based approaches for deterministic robot systems is not feasible. The planner for a deterministic system needs to consider only the set of points that are sampled during the planning process and the points traversed while connecting the sample points. This allows a straightforward

reduction of the state space to a subset that the robot, under the plan, will not leave. In contrast, a stochastic system will have positive (albeit small) probability of reaching a much larger portion of the state space. Thus, if we can only analyze a certain portion of the space due to limited computation, we cannot guarantee (with probability one) that, during execution of the plan, the robot will not leave the set of points we have analyzed explicitly.

Classically, this problem has been handled by optimizing the policy for every point of domain over which the policy is defined. However, as the size and dimension of that domain increase, an overwhelming computational burden is placed upon the planning algorithm. We can utilize control policies that maximize (or minimize) the likelihood of certain events, but for typical robot problems one cannot find policies that localize all the probability density on a target state (or within an arbitrarily small target set of states) and that have zero probability of collisions with workspace obstacles. Despite these difficulties, a large number of approaches have been proposed for the stochastic robot navigation problem. In the next section, we survey those methods.

7.1.2 Stochastic Navigation Methods

There has been a large volume of research on optimizing systems modeled as POMDPs. Some work is directed to specific robotics problems, while other methods focus on computing optimal policies for general systems modeled with stochastic difference equations. Consideration of uncertainty, created by an uncertain process model, was first combined with sampling-methods in [62] to predict the behavior of a system. Rather than just predicting, in [49, 63, 51], attempts were made to extend sampling-based algorithms to plan for systems with uncertainty. The work in [49] considers uncertainty in the robot's motion model, while [63, 51] consider uncertainty in the map of the environment. The Sampling Hyperbelief Optimization Technique [50] also constructs a graph-based representation of trajectories for uncertain systems. However, it differs from these other methods by operating in the space of probability functions over the belief space.

Other work in the robotics community generates plans to minimize uncertainty for specific robot tasks. For example, the active localization algorithms

of [64] and [65] make robot localization more effective by specifically considering expected uncertainty of the localization algorithm while planning the next control the robot will receive. These algorithms generate a control to minimize uncertainty at the next stage, but unlike this work, do not optimize over a path or specify a goal position.

The Belief Roadmap planner (BRM) [48, 66, 48] and the method of [67] have similar goals to our work. Like [48], we use a sampling-based planner, e.g., [60], in the robot's workspace to heuristically construct a set of local policies. However, [48, 66, 48, 67] restrict the set of possible policies to make the optimization procedure feasible and commit to a particular representation of belief (Gaussian). These methods use extended and unscented Kalman filters to approximate the representation of belief as a multi-variate normal distribution over the state space. We differ by using a sequential Monte Carlo representation that can more closely model the true pdf, if a sufficient number of particles are used. Furthermore, the above methods control the growth of possible future belief states by considering only the ML observation when evaluating policies. The main advantage of our method is that we explicitly attempt to characterize many possible future observations at every stage, and can in some cases better gauge the costs different policies will incur. Similar approaches can be used in a forward-based, diffusive search of the reachable belief space, and several approaches using this design principle have been put forth, e.g., [68, 55, 53, 54]. We draw significant inspiration from [28]. In particular, [28] uses similar techniques for prediction of the hyperbelief state, but our optimization approach, search heuristics, and problem goal differ from this work.

Hierarchical approaches for uncertain systems have been explored by several researchers. Approaches for MDP's such as [51] and [52], like roadmap methods, are hierarchical. Semi-Markov decision processes like [52] use macro-actions or options, feedback or open-loop policies that persist until a termination condition is met, to temporally abstract the problem to reduce the number of controls considered. Other hierarchical POMDP methods have been explored in the literature. Planning with a pre-defined hierarchy of tasks has been explored in [43] and [44]. Other methods, such as [45, 46, 47], attempt to discover a hierarchy of tasks to use for planning. Our usage of the term *hierarchical* differs from the above by referring to the method we use for planning, not the structure of the POMDP model or an ordered set

of tasks we wish to perform.

7.2 General Approach

Our approach is to use a continuous POMDP robot model and the optimization method of Chapter 4. We propose two methods based on this general approach. The first is based on using a roadmap heuristic and is largely presented as a comparison against other techniques in the literature. The latter is guided by information-theoretic heuristics and is more general.

Although modeling a robot system as a POMDP is not new, it is generally not possible to apply standard POMDP planning algorithms (either exact or sampling-based) to problems of the complexity typical in robotics applications. Because the sequence of future observations is a random vector that is unknown at planning time, the behavior of the system executing a local policy evolves in the space of posterior pdf's over the belief space, which we refer to as the *hyperbelief space*. For systems with nonlinear models or non-additive noise, the belief typically cannot be represented with a fixed and finite number of parameters. Even if these difficulties are not present, discontinuous or non-smooth observation models or control policies will typically prevent the hyperbelief evolution from being expressed in closed form. These conditions are almost always present in robotics systems, e.g., discontinuities in the environment depth map and control strength saturation. We demonstrate a technique that not only handles these difficulties, but also actively exploits them to achieve the planning objective. We will discuss the specific models, belief representations, and local policies along with the respective methods.

7.3 Navigation using a Roadmap Heuristic

We consider a standard model for a mobile robot whose sensing is based on localization beacons. The robot has configuration in $SE(2)$ and a probabilistic motion model. Observations, correlated with state, arise from a set of noisy beacons, where the accuracy and precision of measurements from a beacon improve as the robot moves closer to the position of the beacon. Given start

and goal configuration, the objective is to minimize an uncertainty measure along the path to the goal.

This system can be modeled as a POMDP with continuous state, control, and observation spaces. We demonstrate that by using our method, we can find policies comparable to and, in some cases, better than other existing methods. This is the case even when restrictive assumptions are imposed (e.g., Gaussian noise, no obstacle constraints). In short, our method is better able to evaluate uncertainty in portions of the space where the transition functions exhibit nonlinear behavior and the approximation of the belief as a Gaussian distribution is not faithful to the true pdf.

We first discuss the robot model and how it is simulated for evaluating policies. Because the system has continuous $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{U}$, we cannot finitely parameterize $\mathcal{P}_b$ and, in general, there is no exact representation of belief that does not grow in complexity with the number of stages executed. We represent a belief by a set of particles, an established procedure for approximating a function [7, 9]. With this approximation, we work with a pmf over a finite set of support points from the belief pdf, but do not discretize the state, observation, or control spaces; i.e., the support points are neither fixed nor representative of a region of the state space.

This problem has previously been considered in [48, 66, 67] for the specific case of finding minimum uncertainty paths for mobile robots localizing with beacons. We consider the same scenario, but compute nearly optimal policies using a new approach that models the system as a POMDP whose cost criterion minimizes uncertainty in the robot's position. Since computing optimal policies for this problem is intractable [1], we use a value function approximation method to find a nearly optimal belief-feedback policy. Furthermore, we demonstrate that our approach can handle additional constraints such as obstacles.

7.3.1 Robot and Environment Model

We use the model used in [48, 67], a mobile robot with configuration in $SE(2)$, moving through an environment with multiple beacons that provide increasingly reliable measurements with decreasing distance to the beacon. We denote the robot's state as $x \in SE(2)$, with $x = [p^1, p^2, \theta]$ where $[p^1, p^2] \in$

$\mathbb{R}^2$ is the robot's position and, $\theta \in \mathcal{S}^1$ is the robot's orientation. The robot's transition model is

$$\begin{bmatrix} \boldsymbol{p}_{t+1}^1 \\ \boldsymbol{p}_{t+1}^2 \end{bmatrix} = \begin{bmatrix} p_t^1 \\ p_t^2 \end{bmatrix} + \boldsymbol{D} \begin{bmatrix} \cos(\theta_t + \frac{\boldsymbol{T}}{2}) \\ \sin(\theta_t + \frac{\boldsymbol{T}}{2}) \end{bmatrix} + \boldsymbol{C} \begin{bmatrix} \sin(\theta_t + \frac{\boldsymbol{T}}{2}) \\ \cos(\theta_t + \frac{\boldsymbol{T}}{2}) \end{bmatrix} \tag{7.1}$$

$$\boldsymbol{\theta}_{t+1} = (\theta_t + \boldsymbol{T}) \mod 2\pi \tag{7.2}$$

where $\boldsymbol{D}$, $\boldsymbol{C}$, and $\boldsymbol{T}$ are noisy inputs to the system. The quantity $\boldsymbol{D}$ is the step length the robot is commanded to move forward. The quantity $\boldsymbol{T}$ is the amount the robot is commanded to turn. The quantity $\boldsymbol{C}$ is the amount the robot is commanded to move perpendicular to the main axis of translation. For many types of mobile robots, $\boldsymbol{C}$ will be strictly zero. The random vector $[\boldsymbol{D}, \boldsymbol{C}, \boldsymbol{T}]$ is generated from the control input and process noise, i.e., $[\boldsymbol{D}, \boldsymbol{C}, \boldsymbol{T}]' = u_t + \boldsymbol{n}_t$ where $u_t \in \mathbb{R}^3$ is the control input to the system. The noise $\boldsymbol{n}_t \in \mathbb{R}^3$ is a random vector drawn from a stationary, mean zero normal distribution with a diagonal covariance matrix.

Noisy measurements of the position of the robot come from a collection of distance beacons. At every stage, the robot receives a measurement from every beacon in the workspace, but the quality of the information conveyed varies with the robot's distance from the beacon. Specifically, the i^{th} beacon reports a measurement $\boldsymbol{y}^i$ that is a random variable with distance-dependent bias in mean and distance-dependent variance. We model the dependences of the mean and variance as linear functions, specifically

$$\mu_b^i(d) = \mu_b^m d + \mu_b^b$$
$$\sigma_b^i(d) = \sigma_b^m d + \sigma_b^b$$

where $d = ||[p^1, p^2]' - p_{\text{b}}^i||_2$ and $p_{\text{b}}^i \in \mathbb{R}^2$ is the location of the i^{th} beacon. Thus, the measurement from the i^{th} beacon is modeled as a random variable drawn from $\mathcal{N}(\mu_b^i(d), \sigma_b^i(d))$, a normal distribution with mean $\mu_b^i(d)$ and variance $\sigma_b^i(d)$. An observation $\boldsymbol{y}_t$ consists of a random vector of measurements, where the i^{th} entry is $\boldsymbol{y}^i$.

This is not a linear Gaussian system, and the belief over the state space with respect to the information history will typically not be a normal pdf. In fact, using the approximation of a normal pdf is often poor because, although all disturbances are drawn from normal distributions, the process

model is nonlinear. This will this cause the distance between the belief and any normal distribution to increase as t increases. Although the pdf of measurements is normal along the line extending from the beacon to the robot, when extending the pdf $\mathrm{f}_{\boldsymbol{x}_t|\boldsymbol{y}_t}(x_t|y_t)$ over the workspace, level sets of constant likelihood form circles centered on the beacon. Thus, the observations provide a circular distortion on the belief, particularly when close to beacons. Other nonlinearities, e.g., obstacles in the workspace, compound this problem.

7.3.2 Belief Approximation

We use a sequential Monte Carlo method [7, 9] and approximate b_t as a collection of weighted particles. The particles are propagated through the POMDP's transition function using a particle filter to maintain an approximation of the belief in a manner consistent with the true POMDP belief state. The particle-filtered representation is also a belief, but is not equal to exact belief conditioned on the information state.

We use Algorithm 2.1 to maintain our approximate belief. The probability transition function, i.e., (2.1), can be achieved for a particle $x^{(i)}$ by sampling the random vector $[\boldsymbol{D}, \boldsymbol{C}, \boldsymbol{T}]'$ from a normal distribution with mean u_t and a diagonal covariance matrix with σ_D^2, σ_C^2, and σ_T^2 along the diagonal, and then updating the particle's state according to (7.1)-(7.2) with the sampled random vector. The variances σ_D^2, σ_C^2, and σ_T^2 quantify the uncertainty of motion in the principal, orthogonal, and turn directions of system motion. The input to the system u_t will be generated by a policy which is discussed in Section 7.3.3.

The probability observation function, i.e., (2.2), re-weights every particle in the set according to an observation $\boldsymbol{y}_t$. In the case of planning, future observations are unknown and will be generated by an observation sampling function which is discussed in Section 7.3.4. To re-weight the particle with $x^{(i)}$, we must find the likelihood of $\boldsymbol{y}_t$ conditioned on the state of the system being $x^{(i)}$. For every $x^{(i)}$ in b_t,

1. Compute the distance from the particle state to every beacon where $d(i, j)$ is the distance from particle i to beacon j.

2. Compute the expected value of $\boldsymbol{y}_t$ conditioned on $\boldsymbol{x}_t = x^{(i)}$. Begin

by computing the expected value of the j^{th} component, i.e., for every beacon

$$\mu_b^j = \mu_b^m d(i,j) + \mu_b^b$$

These are combined to generate the expected value for the observation.

$$\mu_y^{(i)} = \mathrm{E}\left[\boldsymbol{y}_t | \boldsymbol{x}_t = x^{(i)}\right] = [\mu_b^1, \mu_b^2, \cdots, \mu_b^N]'$$

3. Re-weight each particle by the likelihood of the value (received measurement) y_t. The pdf of $\boldsymbol{y}_t$ is a multi-variate normal distribution with mean $\mu_y^{(i)}$ and diagonal covariance matrix with the quantity $\sigma_b^m d(i,j) + \sigma_b^b$ on the j^{th} entry of the diagonal.

As a note, there is no need to explicitly ignore beacons "far" from the robot, like some other methods in the literature, because the likelihood of a measurement from a beacon is computed based on a normal distribution, whose variance increases with distance. The measurements from beacons far from the robot create approximately flat, i.e., close to uniform, pdf's. Thus, the re-weighting will have little effect.

7.3.3 Local Policies

The local policies $\psi^{x_g} \in \Psi$ form a parameterized class of local policies that attempt to direct the robot to target configuration $x_g \in \mathcal{X}$. Given a goal position x_g, control u_t is generated from the belief-feedback law, specified component-wise,

$$\begin{aligned}
u_T &= \operatorname{atan2}\left(p_g^2 - \mathrm{E}\left[p_t^2 | \boldsymbol{b}_t = b\right], p_g^1 - \mathrm{E}\left[p_t^1 | \boldsymbol{b}_t = b\right]\right) - \mathrm{E}\left[\theta_t | \boldsymbol{b}_t = b\right] \\
u_D &= \min\left(\left|\left|\left[\mathrm{E}\left[(p_t^1, p_t^2) | \boldsymbol{b}_t = b\right] - (x_g^1, x_g^2)\right|\right|_2, \; u_{D,max}\right) \\
u_C &= 0
\end{aligned}$$

Essentially, this class of policies drives the expected value of the robot's position towards a desired goal position in the workspace. The policy stops when the mean of the belief gets sufficiently close to the target point or after the policy has executed a set number of stages. Thus, our stopping condition

for ψ^{x_g} is

$$a_{\psi^{x_g}}(b,t) = \mathbb{I}\left(\left|\left|\mathrm{E}\left[(x^1,x^2)|\boldsymbol{b}_t = b\right] - (x_g^1, x_g^2)\right|\right|_2 \leq \epsilon_\phi\right) \vee \mathbb{I}\left(t > t_{\max}\right)$$

where t is the number of stages since ψ^{x_g} was first used and $\mathbb{I}$ is the indicator function.

7.3.4 Observation Sampling Function

As discussed throughout this dissertation, observations are not available at planning time, so we must generate them. This section discusses generating samples from the function $\mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{b}_t}$ given the model $\mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{x}_t}$ and our particular belief representation.

Since the components of the random vector $\boldsymbol{y}_t$ are conditionally independent with respect to $\boldsymbol{x}_t$, we will analyze each beacon separately. Consider the i^{th} component of the random vector $\boldsymbol{y}_t$. Using the law of total probability, we can express the pdf in terms of belief.

$$\begin{aligned}\mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{b}_t}(y_t|b_t) &= \int \mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{x}_t}(y_t|x)\,\mathrm{f}_{\boldsymbol{x}_t|\boldsymbol{b}_t}(x|b_t)\,dx \\ &= \int \mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{x}_t}(y_t|x)\,b_t(x)dx\end{aligned}$$

Of course, we do not have the exact belief, so we use an approximation based on $\hat{b}_t$.

$$\begin{aligned}\hat{\mathrm{f}}_{\boldsymbol{y}_t|\boldsymbol{b}_t}(y_t|b_t) &= \int \mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{x}_t}(y_t|x)\sum_{i=1}^{N} w_t^{(i)}\delta\left(x - x_t^{(i)}\right)dx \\ &= \sum_{i=1}^{N}\mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{x}_t}\left(y_t|x_t^{(i)}\right)w_t^{(i)}\end{aligned}$$

We can efficiently sample from this distribution by, for each component of $\boldsymbol{y}_t$, combining two samples from standard distributions (uniform(0,1) and standard normal). We will treat the "true" particle as a latent variable. Then, use the sample from the uniform distribution to sample a value of this variable. We then use the sample from the standard normal distribution to choose a sample from the corresponding $\mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{x}_t}\left(y_t|x_t^{(i)}\right)$. Samples generated by

this procedure are distributed according to $\hat{\mathrm{f}}_{\boldsymbol{y}_t|\boldsymbol{b}_t}(y_t|b_t)$ and thus approximate $\mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{b}_t}(y_t|b_t)$.

7.3.5 Minimum Uncertainty Planning Method

In our method, one of the main points of emphasis is the integration of domain knowledge into the planner. The assumption of problem-specific domain knowledge is particularly applicable in robotics, i.e., often the local structure of nearly optimal trajectories is known. The combination of local policies with hyper-particle filtering allows us to develop temporal abstraction, which can be used to search the reachable belief space in depth and produce effective policies without requiring an impractical amount of computation.

The core idea of this method is to use paths prescribed in the workspace for deterministic robots as a heuristic, but to evaluate the effects and to plan in the belief space using value function approximation. We begin with a graph in the workspace, where each vertex corresponds to a point in $\mathbb{R}^2$. One of these vertices should be located at the mean of the position for the initial belief b_0. We use Algorithm 4.2 to compute a hierarchical policy for this POMDP.

The policy and belief sampling function is implemented as a queue data structure. When a new belief $\hat{b}^i$ is added to $\mathcal{B}$, it is mapped to a vertex in the workspace graph by choosing the vertex whose position in the workspace is closest to the mean of the position of the belief. Then, for every outgoing edge from that vertex, we add the tuple $(\hat{b}^i, \psi^{x_g})$ where x_g is the location of the target vertex. Initially, the queue is populated by adding b_0 to $\mathcal{B}$. When the queue is empty, the algorithm terminates.

This policy choice assumes that, often, the majority of the probability density, when projected onto the workspace, congregates around vertices. In the context of the model we have explored, this assumption remained valid, but it is possible for a system with more stochasticity that this heuristic may produce undesirable results. At that point, heuristics based on data structures built for the configuration space will most likely not produce effective belief-feedback local policies.

New beliefs are added to $\mathcal{B}$ as a result of evaluating expansions. We choose

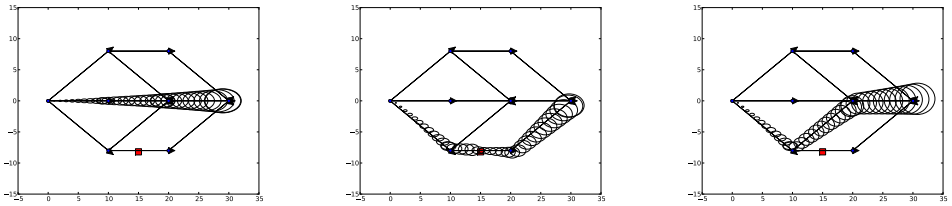
our terminal policy to stop the robot for all future time, i.e., $u_t = 0$ for all $t \geq N$, and compute the terminal cost of stopping at this belief state. If the belief is arbitrarily close to the goal, this will be some finite number that depends on the uncertainty of the belief. Otherwise, it returns infinity to show that this path is not feasible for reaching the goal.

One of the shortcomings of this approach is that we use a single-query planner, despite relying on a planner that does much of its work in pre-processing. While formulated as a single-query planner, the work done by this method is actually much more reusable than one might initially assume. Since we are using the same roadmap and local policies for every query, much of this value function will be already computed if we retain our data structure from the last query. Specifically, the support on which we approximate the value function can largely be reused in subsequent queries. The main source of new computation will be adding policy expansions and beliefs associated with new start and goal vertices on the roadmap. Thus, by saving the value function and augmenting it with every query, we can significantly reduce the amount of computation necessary for repeated queries.

7.3.6 Simulation Results

We tested our results using a mobile robot simulation. After a short discussion of our experimental procedure, we present a very small example that demonstrates how a particle-filtered representation of belief and a POMDP approach is able to improve results over an approximated-Gaussian search of the roadmap. Next, we present larger examples that one might find in a realistic mobile robot scenario.

For purposes of comparison, we evaluated three methods. Method M_1 selects the distance optimal path through the graph, and is provided for a baseline comparison with a plan that does not explicitly consider uncertainty. Method M_2 approximates the belief state as a Gaussian random vector using the extended Kalman filter. Rather than representing a hyperbelief of system evolution, M_2 uses the ML observation at every stage. This produces the same result as the method of [48]. Finally, our proposed method is referred to as M_3. From each method, we generate a belief-feedback policy. Using s sample path simulations, we then are able to evaluate the effectiveness of



(a) Path chosen by M_1 (b) Path chosen by M_2 (c) Nominal path chosen by M_3

Figure 7.1: Approximated Gaussian, ML Observation Estimates of Covariance

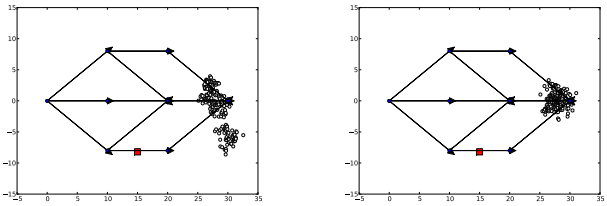
the policies by comparing the mean and variance of the costs of the sample paths. For these results we chose $s = 1000$.

We compared the three methods using a cost function that measures uncertainty at the goal,

$$J_{T_r}(b_0, \pi) = \mathrm{E}\left[\; ||\Sigma_{T_r}||_F^2 \; \middle| b_0 \right]$$

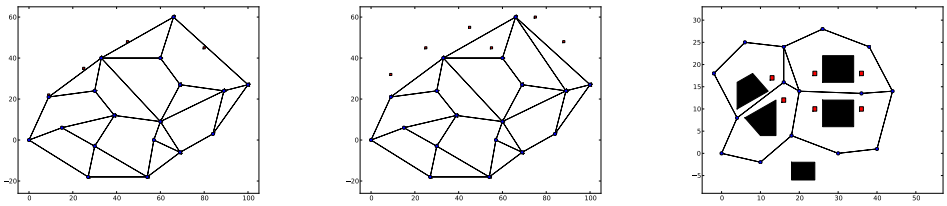
where Σ_t is the sample covariance matrix of $[x^1, x^2]'$ conditioned on $\hat{\boldsymbol{b}}_t$, $||\cdot||_F^2$ denotes the square of the Frobenius norm, which in this case is equal to the sum of squared eigenvalues of the matrix, and T_r is the retirement stage. Many other cost metrics are available, e.g., total uncertainty along the path, average uncertainty along the path, or maximum uncertainty along the path. Multi-objective cost functions, such as those minimizing a combination of distance or time and uncertainty or penalizing failure to reach the goal, can also be used. Furthermore, there are other ways to measure uncertainty, such as entropy-based measures. We chose a metric based on covariance to demonstrate our results because it minimizes a sampled version of the criterion used for optimization by M_2. We have specifically chosen experiments that highlight the differences between these three methods.

Consider the example roadmap pictured in Fig. 7.1. The points represent roadmap vertices and the lines represent edges. The starting belief has the robot beginning at the leftmost vertex, with probability 1. The desired goal is the rightmost vertex. The beacon is represented by the red square. Figure 7.1(a)-(c) show the covariance estimates along the graph paths chosen by M_1, M_2, and M_3, respectively. When close to a beacon, a measurement may have the effect of splitting a uni-modal probability distribution into a multi-



(a) Possible terminal belief using M_2 (b) Possible terminal belief using M_3

Figure 7.2: Notable Belief States



(a) Environment 2 (b) Environment 3 (c) Environment 4

Figure 7.3: Workspaces, Beacons, and Roadmaps for Experiments

modal one. This is demonstrated in Fig. 7.2(a), picturing a final belief from a sample path simulation using the M_2 optimal path. However, taking a path near to the beacon, but sufficiently far to minimize the nonlinear effects of the observation, avoids this difficulty, e.g., Fig. 7.2(b). In fact, M_3 recognizes this during the planning process and decides to (nominally) use the path pictured in Fig. 7.1(c). This improves performance of $\mu_3, \sigma_3^2 = (3.471, 4.054)$ versus $\mu_1, \sigma_1^2 = (4.046, 6.605)$ and $\mu_2, \sigma_2^2 = (4.428, 18.065)$, where μ_i, σ_i^2 are the mean and variance of the costs of the sample path simulations using the policy specified by M_i.

A larger example is shown in Fig. 7.3(a)-(b). These examples highlight the same problem with nonlinearities close to beacons, as in the small example. For the environment of Fig. 7.3 (a), we have, $\mu_3, \sigma_3^2 = (4.921, 14.092)$ versus $\mu_1, \sigma_1^2 = (5.186, 18.029)$ and $\mu_2, \sigma_2^2 = (6.166, 42.275)$. Note that not only is the mean lower for M_3, using M_2 will also result in a higher variance; i.e., the cost along that path will vary more with respect to different sequences of observations.

For the environment of Fig. 7.3 (b), the M_2 nominal path through the graph (the path closest to the beacons) is, in fact, optimal. This is reflected

by the similarity between μ_2 and μ_3, with $\mu_2, \sigma_2^2 = (2.749, 4.278)$ versus $\mu_3, \sigma_3^2 = (2.695, 3.306)$ as compared to $\mu_1, \sigma_1^2 = (3.473, 6.568)$. The mean for M_3 is slightly lower, due to the fact that the policy produced is a true belief-feedback policy. For different belief stages, e.g., belief states with means centered on different sides of a vertex, the policy may choose to take different graph paths. This greater amount of freedom allows the robot to reduce the cost, slightly in this case.

Consider the example environment shown in Fig. 7.3(c), an environment with obstacles. The start configuration is in the lower left corner, and the goal is on the far right. We assume collisions prevent the robot from moving forward, and this causes the Gaussian approximation of belief to be even less useful. In this environment, we evaluate policies not only based on μ_i and σ_i, but also on the probability of failure p_i^{fail} that corresponds to the goal not being achieved in 200 stages. We will discuss two cases with varying levels of process noise.

For both cases, the path through the graph chosen by M_1 and M_2 is the one that weaves through the two narrow passages in the workspace. In the case with very little process noise, M_3 chooses the same nominal path. For this example we have $\mu_3, \sigma_3^2 = (1.984, 1.910)$, $p_3^{\text{fail}} = 0$, as opposed to $\mu_2, \sigma_2^2 = (0.868, 0.199)$ but with $p_2^{\text{fail}} = 0.225$. These results can be explained by M_3 attempting to move the robot between the obstacles, but when the policy becomes more uncertain of being able to navigate the passage it is able to take an alternate route. These alternate routes are more uncertain, so the average uncertainty increases. M_2 does fail frequently, but when it succeeds the results are good. One would expect similar mean and variance for M_3 on the sample paths that successfully navigate between the obstacles.

When we increase the amount of process noise, M_3 recognizes that it is frequently unable to navigate the first narrow passage, and attempting to do so typically causes the uncertainty to increase before taking an alternate route. Thus, M_3 nominally chooses the lower path around the first set of obstacles, then moves up to the middle route to take the second narrow passage, which is closer to the beacons. This results in $\mu_3, \sigma_3^2 = (0.848, 0.206)$, $p_3^{\text{fail}} = 0.088$, as opposed to $\mu_2, \sigma_2^2 = (0.881, 0.370)$, $p_2^{\text{fail}} = 0.203$. The penalty for the robot failing to reach its target can also be increased as a design decision, causing M_3 to nominally take the lower path through the entire graph to the goal.

7.4 Navigation using Information Theory Heuristics

In this section, we present another approach to the minimum uncertainty navigation problem using information-theoretic heuristics. Previously, we discussed the trade-offs inherent in forming a policy to accomplish the minimum uncertainty navigation task. Elements of this trade-off can be captured by incorporating ideas from information theory.[1] Paths that serve to reduce uncertainty can be characterized by having a high expected information gain, where the expectation is computed over the space of possible observations. Paths that minimize the effects of process noise can be characterized by having minimally increasing H, and a control can be chosen to minimize this objective. It is also possible to implement goal-seeking behavior, for example, by minimizing the Kullback-Leibler (KL) divergence between the expected future posterior pdf (i.e., the pdf over the robot's state at some future stage) and a pdf representing the goal. In practice, one can compute controls that achieve one of these objectives over a short time span, but no one of these heuristics is equivalent to our objective. Our approach is to define a set of local policies based on these criteria. We then optimize the transition law of a switched policy whose modes correspond to the local polices to synthesize a composite policy to achieve the specified control objective.

We demonstrate that using information-theoretic heuristics to guide the POMDP optimization process results in a viable motion planning procedure. We additionally use an idea related to regularization kernels [7] in conjunction with a particle filter to compute a continuous approximation of the belief state of the system, which is required by our information-theoretic local policies. We propose what is, to the authors' knowledge, a novel filtering algorithm that tracks collision-free trajectories of a system and simultaneously estimates the probability of trajectories leading to collision.

7.4.1 Problem Definition

We now formally define our minimum uncertainty navigation objective. The quantity $\boldsymbol{p}_t^{\text{fail}}$ denotes the probability the robot collides with an obstacle in the first t stages, and let c^{fail} be a user-defined constant in $[0, +\infty)$ that

[1] The use of information theory in robotics and planning has been explored in many contexts, e.g., [69, 34, 70, 71, 66, 72].

weights the importance of safety from collision versus the eventual distance from the desired goal state. We will quantify that distance using the KL divergence, explicitly defined in (7.4), which measures the dissimilarity between two probability density functions.

The target pdf b_{goal} will be selected based on target positions in the configuration or workspace. This distribution can be specified for a number of nontrivial goal objectives, but we will consider it to simply encode the robot attempting to reach x_{goal}, i.e.,

$$b_{\text{goal}} := \delta(x - x_{\text{goal}})$$

where $\delta(\cdot)$ is the Dirac delta function. The planning objective[2] is then to find a belief-feedback policy π^* that minimizes

$$J(b_0, \pi) = \mathrm{E}\left[c^{\text{fail}} \boldsymbol{p}_T^{\text{fail}} + \mathrm{KL}(\boldsymbol{b}_T || b_{\text{goal}})\right] \tag{7.3}$$

where b_0 is the initial belief of the robot's state and the policy modifies the behavior of the system, i.e., the evolution of $\boldsymbol{b}_t$ and $\boldsymbol{p}_t^{\text{fail}}$. The planning objective is defined in terms of a dynamically chosen (as part of the control law) stopping time T.

7.4.2 Robot Model and Background

The geometry of a robot system can be parameterized by a configuration $q \in \mathcal{Q}$. For a purely kinematic system model, the system's state $x \in \mathcal{X}$ is identical to the configuration. If the system dynamics are also modeled, the state will be $x = [q, \dot{q}]'$. A robot system has a process model

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, u_t, \boldsymbol{n}_t)$$

where $u_t \in \mathcal{U}$ is the control (action) and $\boldsymbol{n}_t$ is a random vector representing process noise which is drawn from an iid random process. This model encodes the kinematic or dynamic trajectory constraints of the robot system, and how

[2]This is a scalarized multi-objective cost function, where the two objectives are often in competition with one another. In principle, any multi-objective optimization characterization could be used in place of scalarization, e.g., computing a Pareto optimal set of policies. Our optimization procedure allows separation of each objective until the value is computed, so it does not preclude other techniques.

the process is affected by uncertainty $\boldsymbol{n}_t$. The observation model dictates how the robot's state affects its sensors, and is modeled

$$\boldsymbol{y}_t = h(\boldsymbol{x}_t, \boldsymbol{m}_t)$$

where $\boldsymbol{m}_t$ represents the sensor noise. For our analysis, we will assume $\mathcal{X}$, $\mathcal{U}$, and $\mathcal{Y}$ are continuous. However, the random vectors $\boldsymbol{y}$, $\boldsymbol{x}$, or $\boldsymbol{u}$ may also be discrete or mixed. The same analysis can be applied in these cases with minor modification.

In the development that follows, we will use a number of concepts from information theory. (A standard reference text on information theory is [73].) The *differential entropy* (which will subsequently be referred to as entropy) is a measure of the uncertainty associated to a random variable

$$\mathrm{H}(\boldsymbol{x}) = -\int_{\mathcal{X}} b_t(x) \log_2 b_t(x) dx$$

The KL divergence is used as a pseudo-metric of distance between two pdf's

$$\mathrm{KL}(b^i || b^j) = \int_{\mathcal{X}} b^i(x) \log \frac{b^i(x)}{b^j(x)} dx \tag{7.4}$$

as is common in the literature.

7.4.3 Local Policies

We use two kinds of local policies: those that are designed to decrease uncertainty in the robot's state, and those that draw the robot toward the goal state. The switched policy that results will have the ability to exploit greedy localization behaviors to reduce uncertainty in state, while using greedy goal-searching behaviors to increase the likelihood of reaching the goal state. Greedy strategies suffer from the well-known problem of local minima, and this remains true when planning in the belief space. Therefore, we also sample target points in the belief space to force the search tree to be expanded throughout the reachable belief space, and thus explore the free configuration space.

To reduce uncertainty in the robot's state, we use the policy that minimizes the next-stage expected entropy. This local policy will proceed until

entropy has reached a local minimum, or a maximum number of stages has elapsed. Let $\psi_{\mathrm{H}} = \{\pi_{\mathrm{H}}(\cdot), a_{\mathrm{H}}(\cdot)\}$ represent the next-stage expected entropy minimizing policy. Thus,

$$\pi_{\mathrm{H}}(b_t) = \arg\min_{u_t \in \mathcal{U}} \mathrm{E}\left[\mathrm{H}(\boldsymbol{b}_{t+1}) | \boldsymbol{b}_t = b_t\right]$$

for all $b_t \in \mathcal{P}_b$. Its stopping condition is

$$a_{\mathrm{H}}(b_t, \Delta t) = \left(\min_{u_t \in \mathcal{U}} \mathrm{E}\left[\mathrm{H}(\boldsymbol{b}_{t+1}) | \boldsymbol{b}_t = b_t\right] > \mathrm{H}(b_t)\right) \vee (\Delta t > t_{\max})$$

which means the policy will continue until entropy has reached a local minimum, or a maximum number of stages has elapsed. The quantity Δt refers to the number of time stages that have elapsed during the execution of the local policy, and $t_{\max}$ is a constant specifying a maximum number of stages that we will allow this policy to proceed under any conditions.

To implement target seeking behavior for both the goal configuration and sampled configurations, we use a policy that minimizes the KL divergence to a sampled target, specified by a Dirac delta function centered on a state space target. We will use this policy to draw the robot towards the goal as well as other sampled target points, to aid in obstacle avoidance. Let $\psi_G(b^g) = \{\pi_G(b^g, \cdot), a_G(b^g, \cdot)\}$ represent a target seeking local policy where b^g is the target point in the belief space the robot attempts to reach. This policy satisfies

$$\pi_G(b^g, b_t) = \arg\min_{u_t \in \mathcal{U}} \mathrm{E}\left[\mathrm{KL}(b^g || \boldsymbol{b}_{t+1}) | \boldsymbol{b}_t = b_t\right]$$

for all $b_t \in \mathcal{P}_b$. Its stopping condition is

$$a_G(b^g, b_t, \Delta t) = \left(\min_{u_t \in \mathcal{U}} \mathrm{E}\left[\mathrm{KL}(b^g || \boldsymbol{b}_{t+1}) | \boldsymbol{b}_t = b_t\right] > \mathrm{KL}(b^g || b_t)\right) \vee (\Delta t > t_{\max})$$

This policy executes until progress can no longer be made, or again, a pre-specified maximum number of stages elapses. Of course, there are difficulties when, for all possible controls, the expected next-stage belief has no support at the target point. This problem can be avoided by discarding samples that are "far" from the current belief or by resorting to a simpler policy, e.g., moving the expected mean of $\boldsymbol{b}_{t+1}$ towards the target point until there is

overlapping support.

In general, these strategies cannot be computed in closed form, but numerical approaches such as Monte Carlo simulation can be applied. In this paper, we have computed the policies using a numerical brute force approach. However, one can often do better by considering the approximated structure of a belief, e.g., a set of particles combined with kernel functions to represent the next-stage expected pdf. An interesting direction for future research will be considering general principles that allow one to exploit this structure, i.e., using a mixture of kernels to approximate information-theoretic heuristic policies efficiently.

7.4.4 Belief Filtering

As usual, we turn to a particle filtering approximation of the belief, and a standard particle filter approximates a function with a finite number of weighted support points. However, for some operations, e.g. computing collision probability, differential entropy, and KL divergence, we require a continuous probability distribution. To reconcile these two techniques, we convert that set of particles to a continuous approximation of the belief using regularization kernels [7]. We use a modified version of a standard particle filter algorithm to track collision-free trajectories of the system, and simultaneously estimate the probability of collision. Thus, we modify the canonical algorithm to estimate probability of collision, given an approximated pdf over the state space, and then resample from the collision-free portion of that pdf. The steps of the method are shown in Algorithm 7.1.

We begin by taking an approximate belief $\hat{b}_t$ as input, which is represented as a set of particles. Each particle $(x_t^{(i)}, w_t^{(i)})$ contains a state and associated weight. The first task is to approximate Bayesian prediction by propagating existing particles through the system's process model and to estimate the probability the system will collide with workspace obstacles. Because the set of particles approximates a continuous distribution, but all the probability mass is located on a finite number of support points, we have observed that integrating this density over a projection onto $\mathcal{Q}_{\text{obst}}$, the subset of the configuration space where collisions occur, is often a poor approximation of the probability of collision unless the process model is sampled with an extremely

Input: $\hat{b}_t := \{(x_t^{(i)}, w_t^{(i)})|_{i=1}^N\}$, u_t, y_t
Output: $\hat{b}_{t+1} := \{(x_{t+1}^{(i)}, w_{t+1}^{(i)}|_{i=1}^N)\}$, p_{collide}

// Propagate process model probability distribution
$\hat{b}_{t+1|t} \leftarrow \emptyset$
foreach $(x_t^{(i)}, w_t^{(i)})$ *in* $\hat{b}_t$ **do**
  $\bar{x}_{t+1|t}^{(i)} \leftarrow f(x_t^{(i)}, u_t, \mathrm{E}[\boldsymbol{n}_t])$
  $\hat{b} \leftarrow \hat{b} \cup \{(\bar{x}_{t+1|t}^{(i)}, w_t^{(i)}\}$
end
// Evaluate collisions and sample process noise
$p_{\text{collide}} \leftarrow 0$, $\hat{b}_{t+1|t} \leftarrow \emptyset$
foreach $(\bar{x}_{t+1|t}^{(i)}, \bar{w}_{t+1|t}^{(i)})$ *in* $\hat{b}$ **do**
  $\Gamma_{\boldsymbol{q}_{t|t+1}}(\cdot) \leftarrow$ projection of $\kappa(\bar{x}_{t+1|t}^{(i)}, u_t, \cdot)$ onto $\mathcal{Q}$
  $c \leftarrow \int_{\mathcal{Q}_{\text{obst}}} \Gamma_{\boldsymbol{q}_{t|t+1}}(q)dq$
  $p_{\text{collide}} \leftarrow p_{\text{collide}} + \bar{w}_{t+1|t}^{(i)} \cdot c$
  $w_{t+1|t}^{(i)} = \bar{w}_{t+1|t}^{(i)} \cdot (1 - c)$
  Sample $x_{t+1|t}^{(i)}$ according to $\kappa(\bar{x}_{t+1|t}^{(i)}, u_t, \cdot)$
  while $x_{t+1|t}^{(i)}$ *is in collision* **do**
    Sample $x_{t+1|t}^{(i)}$ according to $\kappa(\bar{x}_{t+1|t}^{(i)}, u_t, \cdot)$
  end
  $\hat{b}_{t+1|t} \leftarrow \hat{b}_{t+1|t} \cup \{(x_{t+1|t}^{(i)}, w_{t+1|t}^{(i)})\}$
end
// Update using observation model
foreach $(x_{t+1|t}^{(i)}, w_{t+1|t}^{(i)})$ *in* $\hat{b}_{t+1|t}$ **do**
  $w_{t+1|t}^{(i)} \leftarrow w_{t+1|t}^{(i)} \cdot \mathrm{P}\left[y_t | x_{t+1|t}^{(i)}\right]$
end
// Resample to avoid degeneracy
$\hat{b}_{t+1} \leftarrow \emptyset$
for *i = 1:N* **do**
  Sample $x_{t+1}^{(i)}$ from $\hat{b}_{t+1|t}$ w.p. $w_{t+1|t}^{(i)}$
  $\hat{b}_{t+1} \leftarrow \hat{b}_{t+1} \cup \{(x_{t+1}^{(i)}, \frac{1}{N})\}$
end

Algorithm 7.1: Filtering Algorithm

large number of particles (and then condensed down to a number of particles appropriate for filtering). Alternatively, we have had increased success by transforming the particle set to a continuous approximation of the pdf using kernel functions. Given a particle set $\hat{b} := \{(x^{(i)}, w^{(i)})|_{i=1}^{N}\}$, the approximate pdf is

$$\hat{\mathrm{f}}_{\boldsymbol{x}_{t|t+1}}(\hat{b}, u, x) = \sum_{i=1}^{N} \kappa(x^{(i)}, u, x) \cdot w^{(i)}$$

where $\kappa(\overline{x}, u, \cdot)$ is a kernel representing the one-step transition pdf of the system. Ideally, $\kappa(\overline{x}, u, \cdot)$ is the exact pdf of the random vector that is the result of the transformation $f(\overline{x}, u, \boldsymbol{n}_t)$. It is important to note that κ will only meet the requirements of a true regularization kernel under certain fairly general assumptions on the transition probabilities of the robot system model.

We can project $\hat{\mathrm{f}}_{\boldsymbol{x}_{t|t+1}}(b_{t|t+1}, u_t, \cdot)$ onto the configuration space and integrate over $\mathcal{Q}_{\text{obst}}$. In practice, $\hat{\mathrm{f}}_{\boldsymbol{x}_{t|t+1}}(b_{t|t+1}, \cdot)$ is an approximation and the integration will be computed numerically, so we have only an approximate probability of collision. We then re-weight each particle, removing from it the portion of the particle's weight corresponding to configurations in collision with obstacles. We then resample each particle from within the collision-free portion of its kernel function. This leaves us with a predicted estimate of the robot's state from the set of collision-free trajectories. Since we consider any collision to be a failure, it does not make sense to consider trajectories that have already failed. The rest of the filtering procedure is standard.

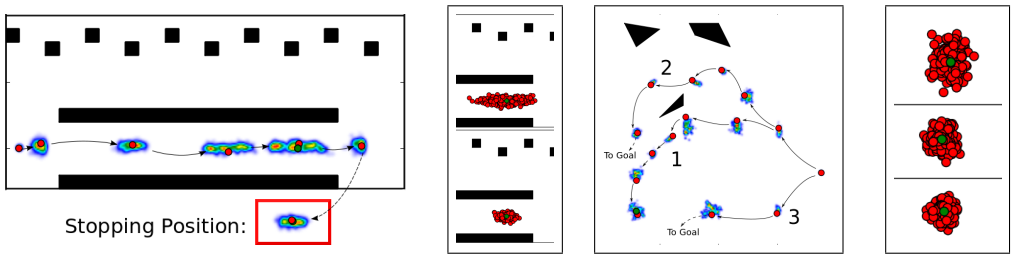
7.4.5 Experiments

We use a simple example to present simulation results that demonstrate the planner is computing intuitively useful policies. We consider a point robot in a planar environment, characterized by the process model

$$x_{t+1} = x_t + u_t + \boldsymbol{n}_t$$

where u_t is bounded, and $\{\boldsymbol{n}_t\}$ is an iid sequence of random vectors drawn from a mean zero, truncated normal distribution. We will consider several nonlinear sensing models that are models of range sensors.

Consider the example environment shown in Figure 7.4(a)-(b). In this simulation, we consider a system with a fixed orientation range sensor, oriented



(a) (b) (c) (d)

Figure 7.4: Plots of Planned Paths and Stopping Points for Sample Path Simulations

along the $+y$ axis (the "up" direction) of the plot. The sensor reports a noisy distance bearing to the closest obstacle. The noise is mean zero and Gaussian, and the variance increases as the distance to the obstacle the robot is measuring increases. Since there is little localization information from the sensor once entering between the two large obstacles, the robot (displayed in red) in Figure 7.4(a) is able to decrease uncertainty by moving past the goal (displayed in green), localizing at the edge of the obstacle, and then returning to the goal.[3] The heat map displayed around the true position of the robot is a plot of the continuous belief approximation (pdf of the robot's state) used at that stage as feedback. Figure 7.4(b) shows a plot of stopping positions for 200 trials using the policy generated by our method (lower plot) versus a plot of 200 trials using the policy that moves directly to the goal. Clearly, the policy generated by our method is significantly better at placing the robot near the goal position. We can quantify this by examining the Frobenius norm of the sample covariance matrix of the stopping position of the trials, which is 0.132 for our method versus 4.19655 for the comparison policy. Another measure of the quality of the method is the average entropy of the filtered belief at the stopping time. Our method produced an average entropy of 7.428 versus 9.080.

We also present a second environment in Figure 7.4(c)-(d). This model used two range sensors that point along the $+x$ and $+y$ axes of the plot. Both sensors have fixed noise variance, but saturate at a maximum distance measurement. Results on this environment are plotted in Figure 7.4(c)-(d)

[3]The final belief state of the robot in Figure 7.4(a) is broken out of the plot so it can be examined separately from the first time the robot passes the goal.

similarly to manner used for Environment 1.

We discuss three types of sample path characteristically generated by three policies evaluated by the planner on a particular optimization trial. The policy labeled 3 in Figure 7.4(c) is the most straightforward route to the goal. However, little localization information is available on this route. The policy labeled 2 moves between the obstacles, and receives the most localization information. The average uncertainty along these paths tends to be smallest. However, the planning algorithm determines that enough information is available localizing on the boundaries of the bottommost obstacle. This in concert with the smaller distance traveled from the last point where a discontinuity in the observation model is present makes the policy labeled 1 the policy our method predicts as optimum. Sample path simulations confirm this, and Figure 7.4(d) shows a plot of stopping positions for 200 trials for each policy, with policy 1 on the bottom, 2 in the middle, and 3 on top. The Frobenius norm of the sample covariance matrix and average entropies for policies 3, 2, and 1 were (0.228, 7.684), (0.0597, 7.382), and (0.026, 7.038), respectively.

Planning times vary significantly based on the environments and system models tested, and less significantly from experiment to experiment mainly due to randomized sampling. The optimization algorithm, executing on a standard desktop workstation, frequently found good policies within a matter of minutes. Though we make no claims about convergence to an optimal policy, the paths shown above were characteristic of paths generated by policies found on nearly all planning runs on these problems, i.e., the algorithm consistently produced paths that demonstrated the same high level behavior.

7.5 Discussion and Future Directions

We have presented two minimum uncertainty planners for stochastic robot models and demonstrated that they are effective, outperforming other methods in the literature. The optimization procedure is expensive and cannot be performed in real time on today's workstation hardware. But, we have demonstrated that good policies can often be computed on the scale of minutes on typical desktop hardware. Thus, although currently our approach leaves something to be desired for real-time application, hardware advances,

potential for parallelized implementation, other implementation quality issues, and likelihood of further algorithmic advancements lead us to believe that this style of method will soon be practical for implementation on many types of systems.

We use a POMDP framework and an optimization routine that performs a coarse optimization using a switched policy. In one case, the local modes of the switched policy are guided by a roadmap heurstic. In the other, the local modes of the switched policy are guided by information-theoretic quantities, which heuristically find paths that may lead to a belief-feedback policy that minimizes a criterion based on the uncertainty of the robot at the goal and probability of collision along the path to the goal.

Although formulated here as a single-query planning method, a multiple-query planning method can also be formulated which could potentially drastically speed up individual planning queries. The data structure built by this method in the belief space is akin to an iterative roadmap construction method such as the Probabilistic Roadmap [60] for a stochastic system. Reusing the data structure and augmenting it with each query holds the potential to rapidly speed up the planning process. This approach in combination with better sampling techniques for waypoint pdf's is an interesting direction for future exploration.

Our goal is to optimize a switching law over the belief space, which is continuous and infinite dimensional. We condense nearby beliefs to a single belief point for the purpose of filling volume within the domain of the switched policy. This approximation works well in most cases, but can be improved by the addition of sensitivity functions along the hyperbelief trajectory. This would essentially convert a computation that provides a point-wise view of the value function to a view of a small piece of the value function.[4] Thus, each simulation of an expansion will then be truly applicable over a volume of the belief space. This process can be thought of as constructing "tubes" around simulated trajectories inside which the system will remain

[4] An alternate approach to computing a continuous approximation to value function using a finite set of points is to take advantage of the structure of the value function (concave, piece-wise linear) and interpolate between α-vectors, e.g., [29, 30, 32, 31]. However, this requires a representative set of α-vectors throughout the belief space for quality approximation. Our goal is to compute the value function extremely sparsely and only analyze trajectories very near to the ones we have simulated, which mitigates the need for many α-vectors.

with high likelihood at execution time. Efficient computation of sensitivity functions and integration into the algorithm is an additional avenue of future investigation.

CHAPTER 8

CONCLUSION

This thesis is about approximating a computationally challenging problem to provide robust control solutions for partially observed stochastic processes. Our approach centers on construction of switched belief-feedback policies, and that approach is chosen based on desired application to systems with interesting dynamics, non-trivial planning solution, many degrees of freedom, high levels of uncertainty, and non-trivial control and observation spaces. For the many reasons discussed in Chapter 1, optimizing POMDPs for systems with these characteristics is expensive. Although many approaches can be applied to this problem, as discussed in Chapter 3, usage of these methods is limited by computational restrictions. In practice, an optimum solution cannot be computed in a reasonable amount of time or with a reasonable amount of computational resources.

Our approach computes a policy that is typically good in practice, but tends not to achieve the optimum. The number of control-belief pairs is typically too large to be evaluated often enough to compute the optimum decision at every belief. Thus, our approach explicitly abandons this approach and we instead use local policies, as discussed in Chapter 4. Our method makes decisions about what local policy should be employed at switching points, and then relies on the local policy to specify controls in response to the current belief. This significantly reduces the computation required and, if the quality of local policies is good, then the reduction of quality of the computed policy (with respect to the optimum policy) may be small. We also sacrifice the quality of the policy at low-likelihood events using Monte Carlo approach to direct computational resources to events commensurate with their importance. The ability of events to affect the total expected outcome, of course, has much to do with how often they are expected to occur.

We demonstrated that this approach enabled us to do well on a number of

discrete problems in Chapter 5, including very large problems. We further demonstrated that our approach was useful for minimum uncertainty robot navigation in Chapter 7. We believe our good results on these problems indicate that we have found a decomposition of the POMDP optimization problem that abstracts away some of the "simple" parts of the optimization, while preserving (and focusing effort on) the "hard" parts of the optimization.

Our approach is similar to value iteration, but done in an atypical way. Consider rewriting a model so that switches become individual controls and the expected cost until switch and distribution over switching points becomes one-stage cost and transition probabilities. Our method would simply correspond to performing an anytime version of the value iteration algorithm on this model. Why not simply rewrite the model? Firstly, as demonstrated in Chapter 4, rewriting the model is hard. In fact, we often cannot compute an expansion without approximation. Since we use closed-loop policies and stopping conditions based on more than just time, finding the new one-stage transition probabilities is not as simple as a few matrix multiplications. Thus, asking an engineer to perform this transformation before optimization is likely to degrade solution quality by soliciting a model that has been speciously approximated. For many systems, the model is written at a granularity that is able to both simply and concisely characterize the true system. Thus, our method allows the engineer to focus on what he or she does best, i.e., write an accurate model at the right level of abstraction and a set of quality local policies. The rest is left to the optimization algorithm; i.e., decompose that optimization problem in such a way that quality results can be computed in a reasonable amount of time.

We have learned, not surprisingly, that the structure of the model plays a huge role in the optimization process. Because the belief transition and hyperbelief transition equations arise from Bayesian filtering, these operators preserve smooth regions and discontinuities of the process and observation models. We believe this is one of the main reasons that our problem decomposition works so well: at a large portion of the time steps, the policy does not have to make complicated decisions. Although currently this only motivates our approach, in the future we will look to exploit this transformation explicitly.

This dissertation discusses, at length, the lifting process by which we can consider our model a POMDP, MDP, or nonlinear system with respect to

the choice of "state" variable. While these transformations are known among those who study this area, we believe it is an important way to characterize the problem and a useful paradigm for algorithm construction.

We consider the work in this thesis to be a prototype of a technique that can be used in many ways for learning and optimization. Although we have demonstrated some promising results on a few problems, we believe that there is still significant progress to be made along this line of research. In the next section, we outline a few directions in which a new step can be taken.

8.1 Future Work

In this section, we outline a few directions in which immediate progress can be made from the work in this dissertation. The discussion in this section is roughly organized from specific improvements that could be made immediately to ideas generated from this work that could eventually be explored.

Although the method we have presented is computationally feasible, there are many algorithm efficiency issues to be addressed that could significantly improve the approach. The two most obvious improvements would address parallelization of expansion approximation and accelerating the expensive nearest neighbor procedure. As mentioned earlier, the nearest neighbor procedure in a high-dimensional space, required at every iteration, is a serious computational bottleneck for our method. Because of dimensionality and the number of stored beliefs, many tree-based data structures one would immediately consider are typically not better than a brute force approach. Future directions could potentially include approximate nearest-neighbor algorithms, space-based hashing of beliefs, or efficient approximations of belief distance. We have implemented a rudimentary approach involving projection of beliefs onto moments. This lower dimensional space (of moments) can then be used to approximately compute nearest neighbors with a significant speed increase. However, much more work needs to be done to explore best practices in terms of both efficiency and robustness.

With multi-core and cluster computing becoming ubiquitous, parallelization is a topic worth exploring for nearly all algorithmic work. While our algorithm is specified as a serial sequence of operations, it can easily be

made parallel by sampling multiple expansions at each iteration. This approach is natural as the sampling and resulting computations are a piece of an incremental computation, and the graph optimization represents the best solution over the partial computation we have performed. Again, we have implemented a rudimentary parallel implementation of this work and the decrease in the time necessary to compute a solution (that performed equally with the result of the serial algorithm) was significant.

Of course, with any sampling-based approach the sampling strategy plays a large role. Our hypothesis is that for this problem, beliefs should receive (probabilistically) a fraction of computation that is roughly commensurate with their likelihood of being reached; i.e., we should allocate more planning to situations commonly encountered.[1] A Monte Carlo walk of the graph from the initial belief with some stopping probability at every step can provide a simple, workable sampling strategy. Typically, one can do better by utilizing strategies that require a ranking of the importance over all beliefs in the graph. Producing this ranking is an area of significant importance, as the ranking determines the sampling, which greatly influences the speed and effectiveness of the algorithm. We have developed a basic importance sampling procedure based on a PageRank-like [74] algorithm, which is discussed in Appendix F. However, this is just an initial approach to a problem that requires significantly more study.

Our current approach has largely been based on developing algorithms that use approximations of the belief and hyperbelief pdf's. This is because, under the lifting process by which we can consider our model a POMDP, MDP, or nonlinear system with respect to the choice of "state" variable, exact pdf's tend to be difficult to represent due to the interaction of complicated system models with the lifting equations. An alternate approach is identification of specific classes of systems that behave "nicely" under the lifting transformation. Thus, rather than only investigating approximation methods, it makes sense to identify classes of models that can be optimized with little computational expense. These models could potentially be used as approximations to similar models that require a more complicated optimization procedure. This is, of course, a standard engineering process. But, by performing this type of analysis with specific attention given to the lifting

[1] This tradeoff should also depend on the cost decrease that is expected from additional planning, but we will ignore this detail in this high-level discussion.

process, we postulate that we can develop effective solutions for a number of hard POMDP problems.

One such example is the probabilistic search problem where the probability of sensing the object is a function of distance from the object. Approximating the initial belief and process and observation models within the exponential family of distributions, under certain conditions, we can guarantee the pdf over the location of the object will remain within a class of a mixture of exponential functions, and the hyperbelief can be represented with a bounded number of parameters. Because we do not immediately have to turn to approximation techniques for such systems, we have the capability to replace many of the tasks that were performed numerically with closed-form, significantly more efficient replacements.

Of course, simple parameterizations for belief and hyperbelief will typically not be possible. However, considering the structure of problems, we can often develop a hybrid belief representation that is faithful to the true belief, but significantly more efficient (and allowing significantly more non-numerical analysis) than blindly applying a particle filter. For example, in the robot navigation problems presented in Chapter 7, the belief could often be approximated very well with a normal distribution. Unfortunately, as we showed, this approximation is not always appropriate and using it exclusively can degrade the solution quality of the optimization procedure. It may be possible to create a system by which the belief approximation is parameterized differently in different parts of the belief space. Development of such a hybrid belief representation goes hand in hand with exploration of the lifting transformation.

The work in this dissertation bears significant resemblance to event-based planning; i.e., the system should behave in a certain way until some event occurs. While we considered local policies to be fixed in this work, an interesting direction of future work involves identifying what are the appropriate events at which a switch should occur. We believe significant gains can be made by choosing events that correspond to crossing or being near to discontinuities in the process and observation models. By choosing events appropriately, we can most likely simplify belief representation between events, as previously discussed, by a single mode of a hybrid belief representation.

Finally, this work was targeted mainly at robot motion planning. We have made some progress on the minimum uncertainty navigation problem, but

significant additional progress still needs to be made before an algorithm that can easily be used in practice emerges. By exploring belief representations based on projection onto exponential families, sampling issues, and development of sensitivity regions for continuous problems, we hope to soon develop a practical and effective stochastic motion planning algorithm.

APPENDIX A

DERIVATION OF THE BELIEF TRANSITION OPERATOR

We begin deriving the belief transition operator, i.e., the Bayesian filtering equations for a POMDP, with the definition of the belief.

$$\begin{aligned} b_t(x_t) &:= \mathrm{f}_{\boldsymbol{x}_t|\boldsymbol{I}_t}\left(x_t|I_t\right) \\ &= \mathrm{f}_{\boldsymbol{x}_t|\boldsymbol{y}_t,u_t,I_{t-1}}\left(x_t|y_t,u_t,I_{t-1}\right) \end{aligned} \tag{A.1}$$

The pdf in (A.1) is identical to the definition of the belief because we simply remove the most recent control and observation from the information vector. Applying Bayes rule, we have

$$b_t(x) = \frac{\mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{x}_t,u_t,I_{t-1}}\left(y_t|x_t,u_t,I_{t-1}\right)\mathrm{f}_{\boldsymbol{x}_t|u_t,I_{t-1}}\left(x_t|u_t,I_{t-1}\right)}{\mathrm{f}_{\boldsymbol{y}_t|u_t,I_{t-1}}\left(y_t|I_{t-1}\right)}$$

which can be simplified by removing the conditioning that provides no information.

$$b_t(x) = \frac{\mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{x}_t}\left(y_t|x_t\right)\mathrm{f}_{\boldsymbol{x}_t|I_{t-1}}\left(x_t|I_{t-1}\right)}{\mathrm{f}_{\boldsymbol{y}_t}\left(y_t\right)}$$

We then rewrite the conditional pdf over x_t using the total law of probability.

$$b_t(x) = \frac{\mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{x}_t}\left(y_t|x_t\right)}{\mathrm{f}_{\boldsymbol{y}_t}\left(y_t\right)}\int \mathrm{f}_{\boldsymbol{x}_t|\boldsymbol{x}_{t-1},u_{t-1}}\left(x_t|x_{t-1},u_{t-1}\right)\mathrm{f}_{\boldsymbol{x}_{t-1}|I_{t-1}}\left(x_{t-1}|I_{t-1}\right)dx_{t-1}$$

The final step is accomplished by noticing that the last factor in the integrand is the definition of the belief at stage $t-1$.

$$b_t(x) = \frac{\mathrm{f}_{\boldsymbol{y}_t|\boldsymbol{x}_t}\left(y_t|x_t\right)}{\mathrm{f}_{\boldsymbol{y}_t}\left(y_t\right)}\int \mathrm{f}_{\boldsymbol{x}_t|\boldsymbol{x}_{t-1},u_{t-1}}\left(x_t|x_{t-1},u_{t-1}\right)b(x_{t-1})dx_{t-1}$$

Thus, we have derived the belief transition operator.

APPENDIX B

DERIVATION OF HYPERBELIEF TRANSITION FUNCTION

In this appendix, we derive (2.5). By definition, the hyperbelief transfer function Φ evaluated at b_{k+1} is given by

$$(\Phi_\pi \beta_t)(b_{t+1}) = \mathrm{f}_{\boldsymbol{b}_{t+1}|b_0,\pi}\left(b_{t+1}|b_0,\pi\right)$$

To obtain the hyperbelief value at stage $t+1$, we marginalize over the stage t beliefs

$$\begin{aligned}\mathrm{f}_{\boldsymbol{b}_{t+1}|b_0,\pi}\left(b_{t+1}|b_0,\pi\right) &= \int_{\mathcal{P}_b} \mathrm{f}_{\boldsymbol{b}_{t+1}|\boldsymbol{b}_t,b_0,\pi}\left(b_{t+1}|b_t,b_0,\pi\right)\mathrm{f}_{\boldsymbol{b}_t|b_0,\pi}\left(b_t|b_0,\pi\right)db_t \\ &= \int_{\mathcal{P}_b} \mathrm{f}_{\boldsymbol{b}_{t+1}|\boldsymbol{b}_t,\pi}\left(b_{t+1}|b_t,\pi\right)\beta_t(b_t)db_t \end{aligned} \tag{B.1}$$

We obtain (B.1) from the fact that the belief at stage $t+1$ is conditionally independent of b_0 given b_t, and by applying the definition of a hyperbelief. We can express $\mathrm{f}_{\boldsymbol{b}_{t+1}|\boldsymbol{b}_t,\pi}\left(b_{t+1}|b_t,\pi\right)$ by marginalizing over $\mathcal{Y}$, i.e.,

$$\mathrm{f}_{\boldsymbol{b}_{t+1}|\boldsymbol{b}_t,\pi}\left(b_{t+1}|b_t,\pi\right) = \int_{\mathcal{Y}} \mathrm{f}_{\boldsymbol{b}_{t+1}|\boldsymbol{y}_{t+1},\boldsymbol{b}_t,\pi}\left(b_{t+1}|y_{t+1},b_t,\pi\right)\mathrm{f}_{\boldsymbol{y}_{t+1}|\boldsymbol{b}_t,\pi}\left(y_{t+1}|b_t,\pi\right)dy_{t+1} \tag{B.2}$$

If $\mathcal{Y}$ is discrete, the integral can be replaced by a summation. Since the source of uncertainty in the belief transition function is the unknown observation, the pdf of the belief transition conditioned on the observation will be a Dirac delta function, i.e.,

$$\mathrm{f}_{\boldsymbol{b}_{t+1}|\boldsymbol{y}_{t+1},\boldsymbol{b}_t,\pi}\left(b_{t+1}|y_{t+1},b_t,\pi\right) = \delta(b_{t+1} - \phi_{y_{t+1},\pi(b_t)}b_t) \tag{B.3}$$

Substituting (B.2) and (B.3) into (B.1), we get (2.5).

APPENDIX C

DERIVATION OF SAMPLE PATH BELIEF PERTURBATION EQUATIONS

We begin by deriving the multi-stage belief transition function. Starting from b_0 and using the belief transition operator (2.4), we have

$$b_1 = \frac{\bar{\phi}_{y_1,u_1} b_0}{\mathbf{1}'\bar{\phi}_{y_1,u_1} b_0} := \eta_{y_1,u_1}\bar{\phi}_{y_1,u_1} b_0$$

Equation (2.4) can be applied again

$$\begin{aligned} b_2 &= \frac{\bar{\phi}_{y_2,u_2} b_1}{\mathbf{1}'\bar{\phi}_{y_2,u_2} b_1} \\ &= \frac{\bar{\phi}_{y_2,u_2}\eta_{y_1,u_1}\bar{\phi}_{y_1,u_1} b_0}{\mathbf{1}'\bar{\phi}_{y_2,u_2}\eta_{y_1,u_1}\bar{\phi}_{y_1,u_1} b_0} \end{aligned}$$

and since η is a scalar, we get

$$b_2 = \frac{\bar{\phi}_{0:2} b_0}{\mathbf{1}'\bar{\phi}_{0:2} b_0}$$

This procedure can be repeated iteratively to get a multi-stage belief transition function for t stages. To analyze the sensitivity of this operator, we insert $b_0 + \Delta b_0$, nominal plus perturbation, into the multi-stage belief transition function for t stages.

$$b_t + \Delta b_t = \frac{\bar{\phi}_{0:t}(b_0 + \Delta b_0)}{\mathbf{1}'\bar{\phi}_{0:t}(b_0 + \Delta b_0)}$$

Then, rearrange to isolate Δb_t.

$$\begin{aligned}
\Delta b_t &= \frac{\bar{\phi}_{0:t}}{\mathbf{1}'\bar{\phi}_{0:t}(b_0+\Delta b_0)}(b_0+\Delta b_0) - \frac{\bar{\phi}_{0:t}b_0}{\mathbf{1}'\bar{\phi}_{0:t}b_0} \\
&= \frac{\bar{\phi}_{0:t}b_0}{\mathbf{1}'\bar{\phi}_{0:t}(b_0+\Delta b_0)} - \frac{\bar{\phi}_{0:t}b_0}{\mathbf{1}'\bar{\phi}_{0:t}b_0} + \frac{\bar{\phi}_{0:t}\Delta b_0}{\mathbf{1}'\bar{\phi}_{0:t}(b_0+\Delta b_0)} \\
&= \frac{[\mathbf{1}'\bar{\phi}_{0:t}b_0]\bar{\phi}_{0:t}b_0 - [\mathbf{1}'\bar{\phi}_{0:t}b_0]\bar{\phi}_{0:t}b_0 - [\mathbf{1}'\bar{\phi}_{0:t}\Delta b_0]\bar{\phi}_{0:t}b_0}{(\mathbf{1}'\bar{\phi}_{0:t}b_0)^2 + (\mathbf{1}'\bar{\phi}_{0:t}b_0)(\mathbf{1}'\bar{\phi}_{0:t}\Delta b_0)} + \frac{\bar{\phi}_{0:t}\Delta b_0}{\mathbf{1}'\bar{\phi}_{0:t}(b_0+\Delta b_0)} \\
&= \frac{[\mathbf{1}'\bar{\phi}_{0:t}b_0]\bar{\phi}_{0:t}\Delta b_0 - [\mathbf{1}'\bar{\phi}_{0:t}\Delta b_0]\bar{\phi}_{0:t}b_0}{(\mathbf{1}'\bar{\phi}_{0:t}b_0)^2 + (\mathbf{1}'\bar{\phi}_{0:t}b_0)(\mathbf{1}'\bar{\phi}_{0:t}\Delta b_0)} \\
&= \frac{[\mathbf{1}'\bar{\phi}_{0:t}b_0]\bar{\phi}_{0:t}\left[I - \frac{\bar{\phi}_{0:t}b_0\mathbf{1}'\bar{\phi}_{0:t}}{\mathbf{1}'\bar{\phi}_{0:t}b_0}\right]\Delta b_0}{(\mathbf{1}'\bar{\phi}_{0:t}b_0)^2 + (\mathbf{1}'\bar{\phi}_{0:t}b_0)(\mathbf{1}'\bar{\phi}_{0:t}\Delta b_0)} \\
&= \frac{\bar{\phi}_{0:t}\left(I - \frac{\bar{\phi}_{0:t}b_0\mathbf{1}'\bar{\phi}_{0:t}}{\mathbf{1}'\bar{\phi}_{0:t}b_0}\right)}{\mathbf{1}'\bar{\phi}_{0:t}b_0 + \mathbf{1}'\bar{\phi}_{0:t}\Delta b_0}\Delta b_0
\end{aligned}$$

This gives Δb_t as a function of Δb_0 and b_0.

APPENDIX D

DERIVATION OF SAMPLE PATH COST PERTURBATION EQUATIONS

For expositional purposes, we begin by defining

$$\begin{aligned}\eta_t &:= \mathbf{1}'\bar{\phi}_{0:t}b_0 \\ \theta_t^n &:= l(u)'\bar{\phi}_{0:t}\left(I - \frac{\bar{\phi}_{0:t}b_0\mathbf{1}'\bar{\phi}_{0:t}}{\mathbf{1}'\bar{\phi}_{0:t}b_0}\right) \\ \theta_t^d &:= \mathbf{1}'\bar{\phi}_{0:t}\end{aligned}$$

Then,

$$g_t(b_0, \Delta b_0) := l(u)'\Delta b_t = l(u)'\frac{\bar{\phi}_{0:t}\left(I - \frac{\bar{\phi}_{0:t}b_0\mathbf{1}'\bar{\phi}_{0:t}}{\mathbf{1}'\bar{\phi}_{0:t}b_0}\right)\Delta b_0}{\mathbf{1}'\bar{\phi}_{0:t}(b_0 + \Delta b_0)} = \frac{\theta_t^n \Delta b_0}{\eta_t + \theta_t^d \Delta b_0}$$

The quantities θ_t^n and θ_t^d are row vectors and η_t is a scalar. The function $g_t(b_0, \Delta b_0)$ is the stage cost at stage t. To form the Taylor approximation, we first compute the gradient[1] of $g_t(b_0, \cdot)$ with respect to Δb_0.

$$\nabla' g_t(b_0, \Delta b_0) = \frac{(\eta_t + \theta_t^d \Delta b_0)\theta_t^n - (\theta_t^n \Delta b_0)\theta_t^d}{(\eta_t + \theta_t^d \Delta b_0)^2} \tag{D.1}$$

Applying the gradient operator to (D.1) we are able to compute the Hessian.

To approximate cost under small perturbations, we form the second order Taylor approximation around b_0, i.e., $\Delta b_0 = 0$. Thus,

$$\hat{g}_t(b_0, \Delta b_0) = g(b_0, 0) + \nabla' g_t(b_0, 0)\Delta b_0 + \frac{1}{2}\Delta b_0' \nabla^2 g_t(b_0, 0)\Delta b_0 + O(||\Delta b_0||^3)$$

[1] We will omit the usual subscript notation, in this case b_0, on the gradient operator ∇ for the remainder of this appendix.

and we can compute $\nabla' g_t(b_0, 0)$ and $\nabla^2 g_t(b_0, 0)$ from (D.1) and its gradient.

$$\nabla' g_t(b_0, 0) = \frac{\theta_t^n}{\eta_t} = l(u)' \frac{\bar{\phi}_{0:t} \left(I - \frac{\bar{\phi}_{0:t} b_0 \mathbf{1}' \bar{\phi}_{0:t}}{\mathbf{1}' \bar{\phi}_{0:t} b_0} \right)}{\mathbf{1}' \bar{\phi}_{0:t} b_0}$$

Once we have these terms, we can substitute them into perturbed sample path cost equation

$$\begin{aligned}
\hat{j}_{\bar{t}}(b_0, \Delta b_0) &= j_{\bar{t}}(b_0) + \Delta j_{\bar{t}}(b_0, \Delta b_0) \\
&= j_{\bar{t}}(b_0) + \sum_{t=0}^{\bar{t}} \gamma^t \left[\nabla' g_t(b_0) \Delta b_0 + \frac{1}{2} \Delta b_0' \nabla^2 g_t(b_0) \Delta b_0 \right] + O(||\Delta b_0||^3) \\
&= j_{\bar{t}}(b_0) + \left[\sum_{t=0}^{\bar{t}} \gamma^t \nabla' g_t(b_0) \right] \Delta b_0 + \frac{1}{2} \Delta b_0' \left[\sum_{t=0}^{\bar{t}} \gamma^t \nabla^2 g_t(b_0) \right] \Delta b_0 + O(||\Delta b_0||^3) \\
&= j_{\bar{t}}(b_0) + d(b_0)' \Delta b_0 + \frac{1}{2} \Delta b_0' D(b_0) \Delta b_0 + O(||\Delta b_0||^3)
\end{aligned}$$

where $d(b_0)$ and $D(b_0)$ are a vector and matrix, respectively, whose construction depends only on b_0 (and the POMDP model), and are defined by

$$\begin{aligned}
d(b_0)' &= \nabla' \Delta j_{\bar{t}}(b_0, \Delta b_0) = \sum_{t=0}^{\bar{t}} \gamma^t \nabla' g_t(b_0) \\
D(b_0) &= \nabla^2 \Delta j_{\bar{t}}(b_0, \Delta b_0) = \sum_{t=0}^{\bar{t}} \gamma^t \nabla^2 g_t(b_0)
\end{aligned}$$

These quantities can be computed iteratively as we simulate the system.

$$\begin{aligned}
\bar{\phi}_{0:s} &= \bar{\phi}_{y_s, u_s} \cdot \bar{\phi}_{0:s-1} \\
d^s(b_0)' &= \gamma^s \nabla' g_s(b_0) + d^{s-1}(b_0)' \\
D^s(b_0) &= \gamma^s \nabla^2 g_s(b_0) + D^{s-1}(b_0)
\end{aligned}$$

starting from $d^0(b_0) = \mathbf{0}$, $D^0(b_0) = \mathbf{0}$, and $\phi_{0:0} = I$.

APPENDIX E

EXPANSIONS USING THE TERMINAL POLICY

For finite horizon retirement problems, the expansion can be computed in the standard manner choosing the stopping condition of the terminal policy as the retirement condition for the controller. It may also be useful, for this structure of cost function, to choose the retirement cost at the belief itself, i.e., $V(b, \overline{\pi}) = c_{T_r}(b)$, as an upper bound on cost. Choosing "retire" as the terminal policy can greatly increase the efficiency of the algorithm, but may require the computation to be performed elsewhere if a "final" local policy is required to achieve significant reward.

For discounted infinite horizon cost systems, a "retire" terminal policy cannot be chosen. However, the same computational gains can be made by approximating the cost-to-go using the terminal policy with a heuristic cost estimate. In some cases, a function can be used that exactly computes the cost-to-go without simulating the system. This, of course, depends greatly on the system in question.

In cases where no good heuristic cost estimate can be chosen, the cost-to-go under a terminal policy can be estimated arbitrarily well by simulation of the system. Let

$$l_{\max} := \max_{b \in \mathcal{P}_b,\ u \in \mathcal{U}} l(b, u)$$

be an upper bound on the system stage cost. Then,

$$\begin{aligned} j_\infty(b, \pi_{\text{term}}) &= j_T(b, \pi_{\text{term}}) + \sum_{t=T+1}^{+\infty} \gamma^t l(b_t, u_t) \\ &\leq j_T(b, \pi_{\text{term}}) + \sum_{t=T+1}^{+\infty} \gamma^t l_{\max} \\ &= j_T(b, \pi_{\text{term}}) + \frac{\gamma^T \gamma l_{\max}}{1 - \gamma} \end{aligned}$$

Thus, if we can tolerate error of at most ϵ, then we can guarantee the error

will be no larger if we simulate

$$T = \left\lceil \log\left(\frac{1-\gamma}{\gamma} \cdot \frac{\epsilon}{l_{\max}}\right) \Big/ \log\gamma \right\rceil$$

stages. Since $J_\infty(\cdot)$ is a convex combination of the costs of the individual sample paths, this is the same number of stages needed to bound the expected cost of the hyperbelief evolution.

APPENDIX F

EXPANSION SAMPLING

In this paper, we gave very little discussion to how to sample the domain of the expansion map, but this issue is critical to the speed of learning good policies and is a subject of future investigation. With no a priori information,[1] sampling beliefs from $\mathcal{B}$ should be weighted by how much potential they have to affect the cost-to-go from the initial belief.

We establish weights on the beliefs in $\mathcal{B}$ by viewing the switching locations as a Markov chain. We generate a $|\mathcal{B}| \times |\mathcal{B}|$ matrix M from the switched policy optimization data structure. The matrix M is a left stochastic matrix with one row and column missing, corresponding to transitions to and from a "terminal" state in the Markov chain. Since the transition state is ergodic, we can safely disregard these matrix entries while keeping an accurate probability distribution over the remaining states. We choose a probability distribution over which local policies we will choose at every belief in $\mathcal{B}$; then the entries of M can be computed from the probabilistic evolution of the POMDP stored in the data structure. Additionally, factors can be included in computation of the entries of M to correspond to the number of stages elapsed, which allows estimation of how much reward reaching a belief can contribute.

To compute the weighting for our importance sampling algorithm, let the length $|\mathcal{B}|$ vector v^j correspond to the probability distribution of j^{th} switch. Then, $v^0 = [1, 0, 0, \cdots, 0]$ and

$$v^{j+1} = Mv^j$$

Let length $|\mathcal{B}|$ vector w be an estimate of the importance of beliefs for sam-

[1] For certain applications, it may be useful to utilize a problem-specific sampling approach.

pling. Then, compute w iteratively using the expression

$$w^{j+1} = w^j + (\mathbf{1} - w^j) \circ v^{j+1}$$

where $\circ$ corresponds to the Hadamard product (element-wise multiplication) and $w^j \to w$. Practically speaking, this algorithm is fairly expensive and w should only be re-computed when the data structure changes enough to significantly perturb w.

In cases where there are a finite number of local policies, it is useful to remove weight from beliefs that have been fully expanded, i.e., all policies have been simulated, as post-processing. This is the main reason we perform this computation, as the sampling procedure can otherwise become "trapped" in a set of fully expanded beliefs with high weight. If this is not a concern, then an equivalent set of samples can be drawn by sequentially sampling the Markov chain directly.

REFERENCES

[1] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[2] R. Smallwood and E. Sondik, "The optimal control of partially observable Markov processes over a finite horizon," *Operations Research*, vol. 21, pp. 1071–1088, 1973.

[3] J. Porta, N. Vlassis, M. Spaan, and P. Poupart, "Point-based value iteration for continuous POMDPs," *Journal of Machine Learning Research*, vol. 7, pp. 2329–2367, 2006.

[4] K. J. Astrom, "Optimal control of Markov decision processes with incomplete state estimation," *Journal of Mathematical Analysis and Applications*, vol. 10, pp. 174–205, 1965.

[5] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Second Edition*. Belmont, MA: Athena Scientific, 1995.

[6] W. Lovejoy, "A survey of algorithmic methods for partially observed Markov decision processes," *Annals of Operations Research*, vol. 28, no. 1, pp. 47–65, 1991.

[7] A. Doucet, N. de Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice*. New York, NY: Springer Verlag, 2001.

[8] J. Davidson and S. Hutchinson, "Hyper-particle filtering for stochastic systems," in *IEEE International Conference on Robotics and Automation*, 2008, pp. 2770–2777.

[9] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.

[10] S. Meyn and R. L. Tweedie, *Markov Chains and Stochastic Stability, Second Edition*. New York, NY: Cambridge University Press, 2009.

[11] R. Bellman, "The theory of dynamic programming," *Bulletin of the American Mathematical Society*, vol. 60, no. 6, pp. 503–515, 1996.

[12] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, 2001.

[13] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[14] L. Kaelbling, M. Littman, and A. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, no. 1, pp. 237–285, 1996.

[15] S. P. Meyn, *Control Techniques for Complex Networks*. Cambridge, UK: Cambridge University Press, 2007.

[16] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[17] P. G. Mehta and S. P. Meyn, "Q-learning and Pontryagin's minimum principle," in *IEEE Conference on Decision and Control*, 2009, pp. 3598–3605.

[18] A. S. Manne, "Linear programming and sequential decisions," *Management Science*, vol. 6, no. 3, pp. 259–267, 1960.

[19] H. Cheng, "Algorithms for partially observed Markov decision processes," Ph.D. dissertation, Commerce and Business Administration, University of British Columbia, 1988.

[20] J. Eagle, "The optimal search for a moving target when the search path is constrained," *Operations Research*, vol. 32, no. 5, pp. 1107–1115, 1984.

[21] G. Monahan, "A survey of partially observable Markov decision processes: Theory, models, and algorithms," *Management Science*, vol. 28, pp. 1–16, 1982.

[22] M. Littman, "Algorithms for sequential decision making," Ph.D. dissertation, Brown University, Providence, Rhode Island, 1996.

[23] E. Sondik, "The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs," *Operations Research*, vol. 26, pp. 282–304, 1978.

[24] L. Platzman, "A feasible computational approach to infinite-horizon partially-observed Markov decision problems," Georgia Institute of Technology, Atlanta, GA, Tech. Rep. J-81-2, 1981.

[25] L. Platzman, "Finite-memory estimation and control of finite probabilistic systems," Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1977.

[26] C. White and W. Scherer, "Finite-memory suboptimal design for partially observed Markov decision processes," *Operations Research*, vol. 42, no. 3, pp. 439–455, 1994.

[27] M. Littman, A. Cassandra, and L. Pack Kaelbling, "Learning policies for partially observable environments: Scaling up," in *International Conference on Machine Learning*, 1995, pp. 362–370.

[28] S. Thrun, "Monte Carlo POMDPs," *Advances in Neural Information Processing Systems*, vol. 12, pp. 1064–1070, 2000.

[29] T. Smith and R. Simmons, "Point-based POMDP algorithms: Improved analysis and implementation," in *Proceedings of Uncertainty in Artificial Intelligence*, 2005, pp. 542–555.

[30] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proceedings of the International Joint Conference on Artificial Intelligence*, vol. 18, 2003, pp. 1025–1032.

[31] M. Spaan and N. Vlassis, "Perseus: Randomized point-based value iteration for POMDPs," *Journal of Artificial Intelligence Research*, vol. 24, pp. 195–220, 2005.

[32] H. Kurniawati, D. Hsu, and W. Lee, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Robotics: Science and Systems*, 2008, pp. 65–72.

[33] S. Ross, J. Pineau, S. Paquet, and B. Chaib-Draa, "Online planning algorithms for POMDPs," *Journal of Artificial Intelligence Research*, vol. 32, no. 1, pp. 663–704, 2008.

[34] N. Roy and S. Thrun, "Coastal navigation with mobile robots," *Advances in Neural Information Processing Systems*, vol. 12, no. 12, pp. 1043–1049, 1999.

[35] A. Brooks, A. Makarenko, S. Williams, and H. Durrant-Whyte, "Parametric POMDPs for planning in continuous state spaces," *Robotics and Autonomous Systems*, vol. 54, no. 11, pp. 887–897, 2006.

[36] E. Zhou, M. Fu, and S. Marcus, "Solving continuous-state POMDPs via density projection," *IEEE Transactions on Automatic Control*, vol. 55, no. 5, pp. 1101–1116, 2010.

[37] J. Kakalik, "Optimum policies for partially observable Markov systems," Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA, Tech. Rep. 18, 1965.

[38] J. Eckles, "Optimum replacement of stochastically failing systems." Ph.D. dissertation, Department of Engineering-Economic Systems, Stanford University, Stanford, CA, 1966.

[39] W. Lovejoy, "Computationally feasible bounds for partially observed Markov decision processes," *Operations Research*, vol. 39, no. 1, pp. 162–175, 1991.

[40] N. Roy and G. Gordon, "Exponential family PCA for belief compression in POMDPs," in *Advances in Neural Information Processing Systems*, 2002, pp. 1635–1642.

[41] P. Poupart and C. Boutilier, "Value directed compression of POMDPs," in *Advances in Neural Information Processing Systems*, 2003, pp. 1547–1554.

[42] X. Li, W. Cheung, and J. Liu, "Decomposing large-scale POMDP via belief state analysis," in *IEEE Conference on Intelligent Agent Technology*, 2005, pp. 428–434.

[43] E. Hansen and R. Zhou, "Synthesis of hierarchical finite-state controllers for POMDPs," in *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling*, 2003, pp. 113–122.

[44] J. Pineau, G. Gordon, and S. Thrun, "Policy-contingent abstraction for robust robot control," in *Proceedings of Uncertainty in Artificial Intelligence*, 2003, pp. 477–484.

[45] M. Toussaint, L. Charlin, and P. Poupart, "Hierarchical POMDP controller optimization by likelihood maximization," in *Proceedings of Uncertainty in Artificial Intelligence*, 2008, pp. 55–60.

[46] L. Charlin, P. Poupart, and R. Shioda, "Automated hierarchy discovery for planning in partially observable environments," in *Advances in Neural Information Processing Systems*, 2007, pp. 225–232.

[47] G. Theocharous, K. Murphy, and L. Kaelbling, "Representing hierarchical POMDPs as DBNs for multi-scale robot localization," in *IEEE International Conference on Robotics and Automation*, 2004, pp. 1045–1051.

[48] S. Prentice and N. Roy, "The belief roadmap: Efficient planning in belief space by factoring the covariance," *International Journal of Robotics Research*, vol. 28, no. 11-12, pp. 1448–1465, 2009.

[49] R. Alterovitz, T. Simeon, and K. Goldberg, "The stochastic motion roadmap: A sampling framework for planning with Markov motion uncertainty," in *Robotics: Science and Systems*, 2007, pp. 246–253.

[50] J. Davidson and S. Hutchinson, "A sampling hyperbelief optimization technique for stochastic systems," in *Workshop on the Algorithmic Foundations of Robotics*, 2009, pp. 217–231.

[51] N. Melchior and R. Simmons, "Particle RRT for path planning with uncertainty," in *IEEE International Conference on Robotics and Automation*, 2007, pp. 1617–1624.

[52] R. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1, pp. 181–211, 1999.

[53] R. He, E. Brunskill, and N. Roy, "PUMA: Planning under uncertainty with macro-actions," in *Proceedings of the American Association for Artificial Intelligence*, 2010, pp. 1089–1095.

[54] H. Kurniawati et al., "Motion planning under uncertainty for robotic tasks with long time horizons," *The International Journal of Robotics Research*, vol. 30, no. 3, p. 308, 2011.

[55] S. Candido, J. Davidson, and S. Hutchinson, "Exploiting domain knowledge in planning for uncertain robot systems modeled as POMDPs," in *IEEE International Conference on Robotics and Automation*, Anchorage, AK, USA, May 2010, pp. 3596–3603.

[56] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 2005.

[57] D. Liberzon, *Switching in Systems and Control*. Boston, MA: Birkhäuser, 2003.

[58] M. Hauskrecht, "Incremental methods for computing bounds in partially observable Markov decision processes," in *Proceedings of the American Association for Artificial Intelligence*, 1997, pp. 734–739.

[59] J. Canny, *The Complexity of Robot Motion Planning*. Cambridge, MA: MIT Press, 1988.

[60] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[61] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation*, 2000, pp. 995–1001.

[62] M. Apaydin, D. Brutlag, C. Guestrin, D. Hsu, J. Latombe, and C. Varm, "Stochastic roadmap simulation: An efficient representation and algorithm for analyzing molecular motion," *Journal of Computational Biology*, vol. 10, pp. 257–281, 2003.

[63] P. Missiuro and N. Roy, "Adapting probabilistic roadmaps to handle uncertain maps," in *IEEE International Conference on Robotics and Automation*, 2006, pp. 1261–1267.

[64] W. Burgard, D. Fox, and S. Thrun, "Active mobile robot localization by entropy minimization," in *Proceedings of the Second Euromicro Workshop on Advanced Mobile Robots*, 1997, pp. 155–162.

[65] N. Roy, W. Burgard, D. Fox, and S. Thrun, "Coastal navigation: Mobile robot navigation with uncertainty in dynamic environments," in *IEEE International Conference on Robotics and Automation*, 1999, pp. 35–40.

[66] R. He, S. Prentice, and N. Roy, "Planning in information space for a quadrotor helicopter in a GPS-denied environment," in *IEEE International Conference on Robotics and Automation*, 2008, pp. 1814–1820.

[67] L. Mihaylova, J. De Schutter, and H. Bruyninckx, "A multisine approach for trajectory optimization based on information gain," *Robotics and Autonomous Systems*, vol. 43, no. 4, pp. 231–243, 2003.

[68] R. He, A. Bachrach, and N. Roy, "Efficient planning under uncertainty for a target-tracking micro-aerial vehicle," in *IEEE International Conference on Robotics and Automation*, 2010, pp. 1–8.

[69] G. Chirikjian, *Stochastic Models, Information Theory, and Lie Groups, Volume I: Classical Results and Geometric Methods*. Boston, MA: Birkhauser, 2009.

[70] H. Feder, J. Leonard, and C. Smith, "Adaptive mobile robot navigation and mapping," *International Journal of Robotics Research*, vol. 18, no. 7, p. 650, 1999.

[71] A. Makarenko, S. Williams, F. Bourgault, and H. Durrant-Whyte, "An experiment in integrated exploration," in *IEEE International Conference on Intelligent Robots and Systems*, 2002, pp. 534–539.

[72] M. Chli, "Applying information theory to efficient SLAM," Ph.D. dissertation, Imperial College London, 2009.

[73] T. Cover and J. Thomas, *Elements of Information Theory*. New York, NY: Wiley, 2006.

[74] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web." Computer Science Department, Stanford University, Stanford, CA, Tech. Rep. 1999-0120, 1999.